Logspace Optimisation Problems and their Approximation Properties

Till Tantau
Technische Universität Berlin
Fakultät IV – Elektrotechnik und Informatik
Franklinstraße 28/29, 10587 Berlin, Germany
Fax +49-30-314-73500
tantau@cs.tu-berlin.de

September 4, 2003

Abstract

This paper introduces logspace optimisation problems as analogues of the well-studied polynomial-time optimisation problems. Similarly to them, logspace optimisation problems can have vastly different approximation properties, even though the underlying existence and budget problems have the same computational complexity. Numerous natural problems are presented that exhibit such a varying complexity. They include the shortest path problems for directed graphs, undirected graphs, tournaments, and forests. In order to study the approximability of logspace optimisation problems in a systematic way, polynomial-time approximation classes are transferred to logarithmic space. Appropriate reductions are defined and optimisation problems are presented that are complete for these classes. It is shown that under the assumption $\mathrm{L} \neq \mathrm{NL}$ some natural logspace optimisation problems cannot be approximated with a constant ratio; some can be approximated with a constant ratio, but do not permit a logspace approximation scheme; and some have a logspace approximation scheme, but cannot be solved in logarithmic space. An example of a problem of the latter type is the shortest path problem for tournaments.

1 Introduction

This paper was written because of two misconceptions. While doing research on logspace selectivity [16], I stumbled across the following problem: Given a tournament graph and two vertices, how difficult is to decide whether there exists a path from the first vertex to the second one? (A tournament is a directed graph in which there is exactly one edge between any two different vertices.) Surely, the problem is easy if we are only interested in a polynomial-time algorithm. However, a polynomial-time algorithm is often only the first step towards understanding the complexity of a problem. We may ask whether the problem can also be solved efficiently in parallel or we could study the problem's space complexity (these questions are known to be closely related [9]). In particular, we can ask whether the problem can be solved in logarithmic space. It turned out that the complexity of the tournament reachability problem is very low [19]: A constant depth circuit (an AC^0-circuit) can decide whether a path exists. Thus the problem can be decide in logarithmic space.

My first misconception was that this result 'settled the complexity of the reachability problem for tournaments in a most favourable way'. This misconception was due to the fact that for many

other problems the complexity of the decision problem dictates the complexity of all sorts of related problems. For example, if GAP $\in$ L, then we can also *construct* a path from the source vertex to the target vertex, we can construct the shortest path, and we can decide whether the shortest path is shorter or longer than a certain number; similarly, if HAMILTON-CIRCUIT $\in$ P, we can also construct a Hamiltonian circuit in polynomial time; and so on.

For the tournament reachability problem, the situation turned out to be more subtle. Although we can efficiently decide whether there exists a path from s to t in a tournament G, the algorithm does not actually *construct* such a path. In order to construct one, we might be tempted to apply the same algorithm as for the standard reachability problem: starting from the source vertex, we iteratively move on to the neighbour that has the least distance to the target. However, for this algorithm we need to solve the following problem: Given a tournament G, two vertices s and t, and a number ℓ, does there exist a path from s to t *of length at most* ℓ? Surprisingly, this problem *turns out to be* NL*-complete* and thus we (presumably) cannot use it to construct a path in, say, logarithmic space.

My second misconception was that this completeness result 'settled the complexity of the reachability problem for tournaments in the least favourable way'. After all, finding the smallest path in a tournament cannot be done in logarithmic space, unless L = NL. However, what about finding *any* path instead of the *shortest* one? A second surprise was in store: We *can* find a path from the source to the target in logarithmic space, provided it exists. Indeed, for any factor $r > 1$ we can find a path that is at most r times as long as the shortest path. Thus the problem can be *approximated* in logarithmic space.

In a nutshell, in this paper I attempt to clarify my misconceptions by establishing a framework for the study of logspace optimisation problems. It allows us to compare the complexity of problems in different ways by asking the following questions about the complexity of shortest path problems:

1. How difficult is the *underlying existence problem*? It asks us to decide whether there *exists* a path between the vertices.
2. How difficult is the *underlying budget problem*? It asks us to decide whether the length of the shortest path is smaller than a certain number.
3. How difficult is it to *construct a path* between the vertices?
4. How difficult is it to *construct a path that is approximately* as long as the shortest path?
5. How difficult is it to *construct the shortest path* between the vertices?

For example, let us answer these questions for the shortest path problems for directed graphs, for undirected graphs, and for tournaments. The underlying decision problems have different complexity: The existence problem for directed graphs is NL-complete, for undirected graphs it is SL-complete, and for tournaments it is in AC^0. Most intriguingly, *the underlying budget problems are* NL*-complete*. The last two of them are new examples of NL-complete problems.

Concerning the construction problems, the problems once more exhibit a different behaviour: For none of them we can find the shortest path in logarithmic space, unless L = NL. For directed graphs we even cannot find *any* path in logarithmic space, unless L = NL; for undirected graphs we cannot find any path in logarithmic space, unless L = SL; and for tournaments we *can find* a path in logarithmic space. As mentioned earlier, for tournaments there even exists a *logspace approximation scheme*. As we shall see, there exist further examples of natural optimisation problems, like the 'hot potato problem', whose budget problems are NL-complete, but which have a logspace approximation scheme.

For the development of a theory of logspace optimisation problems that parallels classical polynomial-time optimisation theory I proceed as follows:

1. Two classes NLO and LO are defined. They are analogues of the polynomial-time classes NPO and PO. Their relationship to NL and L is the same as the relationship of NPO and PO to NP and P.
2. Logspace approximation classes are defined. They differ in how well their elements can be approximated by logspace machines. The classes are stretched out between LO and NLO.
3. Numerous optimisation problems from different fields, though mainly from graph theory, are presented and their complexity is pinned down by placing them inside the class framework.
4. Purely structural separation and completeness results for the introduced classes are shown.

Although the focus of this paper is on the complexity of concrete problems and on concrete logspace algorithms, the study of the structural properties uncovers a number of surprising relationships. For example, we shall see that the problem of finding the shortest path in a directed graph is logspace 1-tt-reducible (or 'optimum-reducible') to the problem of finding the shortest path in an undirected graph. In symbols, SP $\leq^{\log}_{\text{1-tt}}$ USP. Both problems are $\leq^{\log}_{\text{1-tt}}$-complete for the class NLO_{pb} of logspace optimisation problems that have polynomially bounded measure. As a surprising corollary we obtain that the function that maps an *undirected* graph to the length of the shortest path between the first and the last vertex is $\leq^{\log}_{\text{1-tt}}$-complete for the classes OptL_{pb} and OptL_{do} introduced in [1].

All classes introduced in this paper contain both minimisation problems and maximisation problems. As is well-recognised in polynomial-time approximation theory, one must be careful when claiming that a problem is complete for classes containing both minimisation and maximisation problems, since reductions between them are seldomly straight-forward. This is especially true if, as in this paper, one is (also) interested in approximation-preserving reductions. For subclasses of NLO such reductions are possible, but it is not clear whether NLO itself has a complete problem with respect to approximation-preserving reductions.

This paper is organised as follows. In Section 2 logspace optimisation classes are introduced. This includes the classes NLO and LO and different logspace approximation classes in between. We also transfer two notions of reducibility between (logspace) approximation problems: logspace-E-reductions and logspace-AP-reductions. In Section 3 numerous logspace optimisation problems are listed and their complexity and approximability are studied. Building on the results obtained in the previous sections, in Section 4 we summarise the structural properties of logspace optimisation and approximation classes. We show that optimal solutions for logspace optimisation problems can computed by AC^1-circuits. In Section 5 we study applications of the results to other areas of complexity theory. We discuss the relationship of logspace optimisation classes to function classes like OptL and study whether parallel queries to the class SL can be simulated by serial queries.

2 Logspace Optimisation Classes

In this section the formal definition of (arbitrary) optimisation problems is reviewed. Restricting the class of these problems to problems whose solution relation can be checked in logspace with one-way access to the solution yields the class of *logspace optimisation problems*. This class, denoted NLO, has properties that parallel the properties of NPO. For example, the budget problem of every problem in NLO is in NL and the budget problems of every problem in NPO is in NP. Likewise, NLO is linked to OptL in exactly the same way as NPO to OptP. Alongside NLO, two variants NLO_{pb} and NLO_{do} are introduced. For these classes special restrictions are imposed on the measure function.

Optimisation problems in NPO can have vastly different properties concerning their approximability by logspace machines. In order to structurise the study of these properties, different logspace

approximation classes are introduced.

Finally, in order to study the approximation structure of the class NLO, appropriate reductions are introduced, namely logspace AP-reductions (*approximation preserving* reductions) and logspace E-reductions (*error preserving* reductions). We also consider truth-table reductions between optimisation problems, but they are not approximation-preserving and are interesting mainly from a structural point of view.

2.1 Optimisation Problems

In the literature two different approaches have been taken to defining optimisation problems. In the more structurally oriented approach of Krentel [11], an optimisation problem is just a function mapping instances to the values of optimal solutions. In this context, the optimisation problem of the clique problem would be a function that maps a graph to the largest clique in this graph. The main advantage of this approach is that optimisation classes are ordinary function classes, which can easily be compared to other complexity classes. The main disadvantage of this approach is that, similar to decision problems, the functions hide the solutions we are actually interested in. In practice, for the clique problem we are neither interested in its underlying decision problem nor in the problem of telling the size of the largest clique in the graph. We are interested in *obtaining* the largest clique in the graph (or, at least, a clique as large as possible).

The second approach defines optimisation problems as much more structured objects. I follow this approach.

Definition 2.1. An *optimisation problem* is a tuple consisting of an *instance set* $I \subseteq \Sigma^*$, a *solution relation* $S \subseteq I \times \Sigma^*$, a *measure function* $m\colon S \to \mathbb{N}^+$, and a *type* $t \in \{\min, \max\}$.

For example, the optimisation problem MAX-CLIQUE is defined as follows: its instance set is the set of (the codes of) all undirected graphs, the solution relation relates graphs to cliques within these graphs, the measure function maps pairs of graphs and cliques to the size of these cliques, and the type is max. The next definition fixes basic notations and terminology.

Definition 2.2. Let $P = (I, S, m, t)$ be an optimisation problem. Let $x \in I$.

1. Let $S(x) := \{y \in \Sigma^* \mid (x, y) \in S\}$ denote the *solutions for* x.
2. Let $m^*(x) := \min\{m(x, y) \mid y \in S(x)\}$, respectively $m^*(x) := \max\{m(x, y) \mid y \in S(x)\}$, denote the *optimal measure for* x. Note that $m^*(x)$ is undefined if $S(x) = \emptyset$.
3. Let $S^*(x) := \{y \in \Sigma^* \mid m(x, y) = m^*(x)\}$ denote the set of *optimal solutions for* x.
4. Let $\mathrm{R}(x, y) := \max\{m(x, y)/m^*(x), m^*(x)/m(x, y)\}$ denote the *performance ratio of the solution* y.
5. The *existence problem* $P_{\exists\mathrm{sol}}$ is the set $\{x \mid S(x) \neq \emptyset\}$.
6. The *budget problems* are the sets $P_{\mathrm{opt}<} := \{(x, z) \mid \exists y.\ m(x, y) < z\}$ and $P_{\mathrm{opt}>} := \{(x, z) \mid \exists y.\ m(x, y) > z\}$.
7. A function $f\colon \Sigma^* \to \Sigma^*$ *produces solutions for* P if for every $x \in P_{\exists\mathrm{sol}}$ we have $f(x) \in S(x)$. It *produces optimal solutions* if for every $x \in P_{\exists\mathrm{sol}}$ we have $f(x) \in S^*(x)$.

2.2 Polynomial-Time and Logspace Optimisation Problems

By restricting the computational complexity of the set I, the relation S, and the function m, different optimisation classes can be defined. The two most widely-studied ones are PO and NPO.

Definition 2.3. An optimisation problem (I, S, m, t) in the class NPO if I is decidable in polynomial time, S is decidable in polynomial time and is polynomially length-bounded, and m is computable in polynomial time. The problem is furthermore in PO if there exists a function in FP that computes optimal solutions for it.

Two special subclasses of these classes are also frequently studied: NPO_{pb} and PO_{pb}. These classes contain problems whose measure function is *polynomially bounded* in the size of the input. Many natural problems fall into this class. For example, MAX-CLIQUE $\in \text{NPO}_{\text{pb}}$ since the size of the largest clique in a graph is bounded by the number of vertices of the graph.

Definition 2.4. Let C be a class of optimisation problems. Then C_{pb} contains all optimisation problems (I, S, m, t) such that there exists a polynomial p with $m(x, y) \leq p(|x|)$ for all $(x, y) \in S$.

Let us now transfer these definitions to logarithmic space.

Definition 2.5. An optimisation problem (I, S, m, t) in the class NLO if

1. I is decidable in logarithmic space,
2. S is decidable in logarithmic space via a machine that reads the alleged solution in a one-way fashion and is polynomially length-bounded, and
3. m is computable in logarithmic space via a machine that reads the solution and writes the output in a one-way fashion.

A problem in NLO is furthermore in LO if there exists a function in FL that produces optimal solutions for it.

The main technical peculiarity of the definition of NLO is the one-way access to the solution tape. This restriction is necessary to ensure that a nondeterministic logspace machine can 'guess' such a solution, see the proof of the following lemma and theorem.

Lemma 2.6. *Let $(I, S, m, t) \in$ NLO. Then there exists a nondeterministic logspace machine M with at most two nondeterministic choices per state such that for every instance $x \in I$ the following holds: If $y \in S(x)$, then y, regarded as a bitstring, is a string of nondeterministic choices that make M accept and output $m(x, y)$ on that path. If $y \notin S(x)$, then y is a string of nondeterministic choices that make M reject. All outputs of M for an instance x have the same length.*

Proof. Let M_S and M_m be the machines that decide S, respectively compute m. Let $\ell(n)$ be a bound on the length of the output produced by m on instances of length n. The machine M works as follows on input x: First, it nondeterministically guesses a number of 0's that it writes on the output tape. Then it starts a simulation of the machines M_S and M_m. It first simulates M_S and then M_m, each time until they attempt to read a bit from the solution tape. The machine then guesses a bit nondeterministically and continues simulating first M_S and then M_m, making them 'believe' that they have read the guessed bit from the (virtual) solution tape. Each simulation stops, when the machine M_S, respectively M_m, attempts to read the second bit from the tape. Once more we guess this bit and continue the simulation. If the simulation reaches a point where M_S accepts and M_m has finished producing its output, we accept and stop, provided the number of 0's written at the beginning and the number of output bits written later add up to $\ell(|x|)$. In all other cases, M rejects. □

Theorem 2.7. *If $P \in$ NLO, then $P_{\exists\text{sol}}, P_{\text{opt}<}, P_{\text{opt}>} \in$ NL. If $P \in$ LO, then $P_{\exists\text{sol}}, P_{\text{opt}<}, P_{\text{opt}>} \in$ L.*

Proof. Let $P \in \mathrm{NLO}$ be a minimisation problem. Since $P_{\exists\mathrm{sol}} \leq_{\mathrm{m}}^{\log} P_{\mathrm{opt}<}$, we only need to show $P_{\mathrm{opt}<} \in \mathrm{NL}$. Let M be the machine from Lemma 2.6. An NL-machine that decides $P_{\mathrm{opt}<}$ works as follows: on input (x, z) it simulates M and every time M attempts to produce an output bit, this bit is instead compared to a bit of z. If M accepts, the simulating machine accepts if the comparison yielded that M's output is smaller than z. Since NL is closed under complement, we also get $P_{\mathrm{opt}>} \in \mathrm{NL}$. For maximisation problems we argue similarly.

The second claim follows easily from the definition. □

Just as in the polynomial-time setting, it also makes sense to study the classes $\mathrm{NLO}_{\mathrm{pb}}$ and $\mathrm{LO}_{\mathrm{pb}}$. There exists an interesting intermediate class $\mathrm{NLO}_{\mathrm{do}}$ that is snuggled between $\mathrm{NLO}_{\mathrm{pb}}$ and NLO. This class does not have a counter-part in the polynomial-time setting.

Definition 2.8. A problem $P \in \mathrm{NLO}$ is in $\mathrm{NLO}_{\mathrm{do}}$ if the machine M_m that computes m has *deterministic output*. This means that M_m writes the first output bit only after the one-way tape that contains the alleged solution has been read completely.

As pointed out in [1], for machines that produce deterministic output, the output depends only on the last configuration reached before the first output bit is produced. Thus such machines can only produce polynomially many different outputs, which shows $\mathrm{NLO}_{\mathrm{pb}} \subsetneq \mathrm{NLO}_{\mathrm{do}} \subsetneq \mathrm{NLO}$. Note that these inclusions are proper inclusions for purely 'syntactic' reasons, not because of any deep complexity-theoretic results.

2.3 Logspace Approximation Classes

In this section the notions of *polynomial-time approximation algorithm*, *polynomial-time approximation scheme*, and *fully polynomial-time approximation scheme* are transferred to logarithmic space.

Definition 2.9. Let P be an optimisation problem and let $r\colon \mathbb{N} \to \mathbb{Q}$ be a function. A function $f\colon I \to \Sigma^*$ is an *r-approximator for P* if it produces solutions for P and $\mathrm{R}\big(x, f(x)\big) \leq r(|x|)$ for all $x \in I$.

Definition 2.10. Let P be an optimisation problem. A function $f\colon I \times \mathbb{N} \to \Sigma^*$ is an *approximation scheme for P* if for all $x \in P_{\exists\mathrm{sol}}$ and all positive integers k we have $f(x,k) \in S(x)$ and $\mathrm{R}\big(x, f(x,k)\big) \leq 1 + 1/k$.

Definition 2.11. Let $P \in \mathrm{NLO}$.

1. $P \in$ exp-ApxLO if there exists an r-approximator in FL for P, where $r(n) = 2^{n^c}$ for some c.
2. $P \in$ poly-ApxLO if there exists an r-approximator in FL for P, where r is some polynomial.
3. $P \in$ ApxLO if there exists an r-approximator in FL for P, where r is a constant.
4. $P \in$ LAS if there exists an approximation scheme f for P such that $f(.,k) \in \mathrm{FL}$ for all k.
5. $P \in$ FLAS if there exists an approximation scheme for P that can be computed in space $O\big(\log k \log |x|\big)$.

The approximation schemes for problems in FLAS are the best schemes we can reasonably hope for. The definition requires that if we wish to attain a ratio of $1 + 1/k$, we may use space at most $O(\log k \log n)$. For problems in $\mathrm{NLO}_{\mathrm{pb}}$ whose budget problems are NL-complete there is only little chance that we can do any better than this. If we could, for example if we had an approximation scheme that needs only space $O(\log^{1/2} k \log n)$, we could decide the budget problem (and hence all

of NL) in space $O(\log^{3/2} n)$. To see this, note that if the optimal solution is known to be an element of $\{1, \ldots, n^d\}$, then a solution of ratio less than $1 + 1/n^d$ is actually optimal.

Approximation means 'coming up with good solutions efficiently'. In particular, for approximable problems we are able to come up with *some* solutions efficiently. The following theorem formalises this observation.

Theorem 2.12. *If* $P \in$ exp-ApxLO*, then* $P_{\exists\text{sol}} \in$ L.

2.4 Reductions between Optimisation Problems

To conclude this section, three reductions between optimisation problems are introduced. They are transfers of the corresponding polynomial-time reductions to logarithmic space. The first reduction, the *error preserving* reduction [10], is technically easier to use and more restrictive than the more general *approximation preserving* reduction introduced in [5]. For completeness results we use error preserving reductions, since for completeness we are interested in reductions that are as restrictive as possible. The approximation preserving reduction is introduced since, at least in the polynomial-time setting, the extra flexibility of this reduction is needed for certain problems that 'should be reducible to one another', but which are not reducible to each other via error preserving reductions. For an overview of polynomial-time reductions used in conjunction with approximation problems see [5].

The third reduction is the *truth-table reduction*. Truth-table reductions are also less restrictive than error preserving reductions, but differently from the approximation preserving reduction: truth-table reductions turn (several) optimal solutions for the problem which we reduce to into an optimal solution for the problem we wish to solve. 'Turning an optimal solution into an optimal solution' is exactly what we do not require for the approximation preserving reduction. A truth-table reduction does not make any promises about how it will map non-optimal solutions. For this reason, it is not really useful in conjunction with questions of approximability, as is well-recognised in the polynomial-time setting. Truth-table reductions are included in this paper since, as we shall see, there are a number of problems that are complete for certain optimisation classes under truth-table reductions, but which do not appear to be complete for these classes under approximation preserving reductions.

The special case of truth-table reductions where we may ask only a single query has different names in the literature. Orponen and Mannila [17] call them *optimisation reductions*. Krentel [11] calls them *metric reductions*, though he reduces optimal measure functions to each other, not whole optimisation problems. In the present paper they will just be called 1-tt-reductions.

Definition 2.13. Let P and P' be optimisation problems. We write $P \leq_{\mathrm{E}}^{\log} P'$ if there exists a triple (f, g, α) with $\alpha \in \mathbb{Q}$ such that

1. $f, g \in$ FL,
2. for all $x \in I$ we have $f(x) \in I'$ and for all $x \in P_{\exists\text{sol}}$ we have $f(x) \in P'_{\exists\text{sol}}$,
3. for all $x \in I$ and $y \in S'\big(f(x)\big)$ we have $g(x, y) \in S(x)$ and

$$\mathrm{R}\big(x, g(x, y)\big) - 1 \leq \alpha\Big(\mathrm{R}'\big(f(x), y\big) - 1\Big).$$

Definition 2.14. Let P and P' be optimisation problems. We write $P \leq_{\mathrm{AP}}^{\log} P'$ if there exists a triple (f, g, α) with $\alpha \in \mathbb{Q}$ such that

1. $f, g \in$ FL,

2. for all $r > 1$ and $x \in I$ we have $f(x, r) \in I'$ and for all $x \in P_{\exists\text{sol}}$ we have $f(x, r) \in P'_{\exists\text{sol}}$,
3. for all $r > 1$, $x \in I$, and $y \in S'\big(f(x, r)\big)$ we have $g(x, y, r) \in S(x)$ and

$$\mathrm{R}'\big(f(x, r), y\big) \leq r \implies \mathrm{R}\big(x, g(x, y, r)\big) - 1 \leq \alpha(r - 1).$$

Definition 2.15. Let P and P' be optimisation problems. We write $P \leq_{\text{tt}}^{\log} P'$ if there exists a logspace oracle Turing machine M that works as follows: On input of an instance $x \in I$ it writes a (coded) query tuple $\langle q_1, \ldots, q_m \rangle$ with $q_i \in I'$ onto an oracle tape. It then enters an oracle query state, which magically changes the contents of the oracle tape into a tuple $\langle y_1, \ldots, y_m \rangle$ such that each y_i is an optimal solution for q_i, provided such a solution exists, and a special symbol otherwise. No matter which optimal solutions are provided, the machine must then output an optimal solution $y \in S^*(x)$, provided $x \in P_{\exists\text{sol}}$.

3 Logspace Optimisation Problems

In this section we study the approximability of a large variety of logspace optimisation problems, mostly of graph problems. For each problem, the problem is first formulated rigorously as an optimisation problem; then its properties are formulated as a theorem and then proved. The problems are organised according to similarity of their definition, nor according to similarity of their approximation properties.

We start with a family of problems related to the question of which vertex in a vertex-labelled graph is the most 'valuable' vertex reachable from a given vertex. We start with these particular problems since they are convenient for proving the first completeness results. For problems considered later, we can establish completeness by reducing an appropriate 'most valuable vertex problem' to them.

The next problems we tackle are shortest path problems, which are perhaps the most intuitive problems in NLO. As we shall see, shortest path problems have vastly different approximation properties for different kinds of graph. Alongside different negative approximability results, two major positive approximation results are established by presenting appropriate approximation algorithms: the shortest path problem for graphs with bounded independence number has a fully logspace approximation scheme; and the shortest path problem for planar-embedded graphs has a logspace poly-approximation algorithm.

Next, we study a family of problems that I call 'hot potato problems'. A group of people is given a hot potato and must pass it around for a given number of rounds, until it has cooled of. Passing the potato from one person to another induces a certain cost (or value) and we wish to minimise (maximise) this cost (value). All variants of this problem are FLAS and we can very precisely state which of its variants are even LO.

Finally, we study a natural problem from automata theory and show that it is among the most difficult problems in NLO.

3.1 Finding the Most Valuable Vertex of a Graph

We begin our investigation with problems that ask us to find a vertex in graph that is reachable from a start vertex and that is 'as valuable as possible'. Variants of this problem are complete for different logspace optimisation classes.

Before we proceed, let us fix some graph-theoretic terminology. A *directed graph* is a pair (V, E) with $E \subseteq V \times V$. An *undirected graph* is a directed graph with a symmetric edge relation. A *walk* is a sequence $(v_1, \ldots, v_\ell)$ of vertices with $\ell \leq |V|$ such that $(v_i, v_{i+1}) \in E$ for all $i \in \{1, \ldots, \ell\}$. A

walk is a *path* if all its vertices are distinct. The *distance function* $\mathrm{d}_G\colon V \times V \to \mathbb{N} \cup \{\infty\}$ maps a pair (s,t) of vertices to the length of the shortest path between them. A *cyclic walk* is a walk $(v_1, \ldots, v_\ell, v_1)$ with $\ell \geq 2$ in which the first vertex is visited exactly twice (at the beginning and at the end) and the second and third vertices are visited exactly once. A cyclic walk is a *cycle* if all but the first and last vertex are distinct.

The difference between walks and paths can be crucial in approximation theory: the budget problem of finding the longest path in a graph is known to be NP-complete and even hard to approximate [8], whereas for walks it is easy. In this paper only *shortest* path problems are considered. For them, the difference is of less importance since we can easily turn a walk containing a vertex several times into a even shorter path by omitting all vertices between the first and last occurrence of every vertex that appears at least twice. For this reason, I call these problems shortest *path* problems: although solutions are walks, the optimal solutions are always paths.

PARTIAL-MOST-VALUABLE-VERTEX (PARTIAL-MVV)

Instances. Directed graph $G = (V, E)$, a vertex $s \in V$, and a partial weight function $w\colon V \to \mathbb{N}^+$.
Solutions. Walk starting at s in G that ends at some vertex t for which $w(t)$ is defined.
Measure. Weight $w(t)$ of vertex at which the walk ends.

Theorem 3.1.

1. PARTIAL-MVV *is* $\leq_{\mathrm{E}}^{\log}$*-complete for* $\mathrm{NLO}_{\mathrm{do}}$.
2. PARTIAL-MVV $\in$ exp-ApxLO *iff* $\mathrm{L} = \mathrm{NL}$.
3. $\text{PARTIAL-MVV}_{\exists\mathrm{sol}}$ *and* $\text{PARTIAL-MVV}_{\mathrm{opt}>}$ *are* $\leq_{\mathrm{m}}^{\log}$*-complete for* NL.

Proof. 1. We first show that PARTIAL-MVV is complete for the class of maximisation problems in $\mathrm{NLO}_{\mathrm{do}}$. Let $P = (I, S, m, \max) \in \mathrm{NLO}_{\mathrm{do}}$ be a maximisation problem. Let M be the machine constructed in Lemma 2.6. Note that M produces its output deterministically, that is, the first output bit will only be produced after all nondeterministic choices have been made. We construct a logspace-E-reduction (f, g, α) from P to PARTIAL-MVV.

The function f takes an instance x as input. It maps this instance to the configuration graph of M on input x. The vertices of this graph are all possible configurations of M on input x (of which there are only polynomially many). There is an edge from a configuration c_1 to a configuration c_2 if M can get from c_1 to c_2 in exactly one step *without writing an output bit*. Thus all steps that would yield an output are suppressed. In order to decide whether we should attach a value to a given vertex (and, if so, which value), we run the following deterministic logspace algorithm: We run M, starting from the configuration encoded by the vertex. If M never reaches a nondeterministic choice during this simulation and if the simulation leads to an accepting configuration, we decide that we will attach an output to the vertex. In order to obtain this output, we rerun M starting from the vertex and attach the produced output to it. The construction ensures $f \in \mathrm{FL}$.

The function g maps solutions for $f(x)$ to solutions for x. A solution for $f(x)$ is a walk p in the configuration graph of M that ends at some configuration that can be deterministically extended to an accepting configuration. The function g maps (x, p) to a bitstring y as follows: We iterate over the vertices on p. All vertices are ignored, except those where a nondeterministic choice is taken. For these vertices, we output a bit that corresponds to this choice. Clearly we have $g \in \mathrm{FL}$.

By construction, the measure $m_{\text{PARTIAL-MVV}}\big(f(x), p\big)$ is exactly $m_P\big(x, g(x,p)\big)$. To see this, note that the output attached to the last vertex of p is exactly the output of M, when its nondeterministic choices are taken according to $g(x,p)$. This output of M is in turn $m_P\big(x, g(x,p)\big)$. Setting $\alpha = 1$, we get $\mathrm{R}_P\big(x, g(x,p)\big) - 1 \leq \alpha\big(\mathrm{R}_{\text{PARTIAL-MVV}}\big(f(x), p\big) - 1\big)$.

We must now show that we can also reduce all minimisation problems in $\mathrm{NLO}_{\mathrm{do}}$ to PARTIAL-MVV. For this, we show that we can reduce the problem PARTIAL-LVV of finding the *least* valuable vertex to PARTIAL-MVV. Since, by the same argument as above, PARTIAL-LVV is $\leq_{\mathrm{E}}^{\log}$-complete for the class of minimisation problems in $\mathrm{NLO}_{\mathrm{do}}$, this suffices to prove completeness of PARTIAL-MVV for all of $\mathrm{NLO}_{\mathrm{do}}$.

For the reduction PARTIAL-LVV $\leq_{\mathrm{E}}^{\log}$ PARTIAL-MVV, let an instance (G, s, w) for the minimisation problem be given. The instance transformation function f maps (G, s, w) to (G, s, w'), where $w'(v) := \lfloor 2^{2\ell}/w(v) \rfloor$ and ℓ is the length of the code of (G, s, w). The function g maps a walk p to the same p. Since division is in FL, see [3], we have $f \in \mathrm{FL}$.

It remains to show that the ratio is sustained. Let $x = (G, s, w)$ be an instance and let p be a walk for $f(x) = (G, s, w')$. Let t be the last vertex on p. Let t^* be a vertex in the reachability component of s such that $w(t^*)$ is minimal. Then $w'(t^*)$ will be maximal.

We distinguish two cases. First, assume $w'(t) = w'(t^*)$. Then $w(t) = w(t^*)$ since $w'(t) \leq 2^\ell$. Thus a solution with an optimal ratio is mapped to a solution with an optimal ratio. Second, assume $w'(t^*) \geq w'(t) + 1$. Then $w'(t^*)/w'(t) - 1 \geq 1/w'(t)$. Setting $\alpha = 2$ we get

$$\begin{aligned}
\mathrm{R}_{\text{PARTIAL-LVV}}\big(x, g(x,p)\big) - 1 = \frac{w(t)}{w(t^*)} - 1 &= \frac{2^{2\ell}/w(t^*)}{2^{2\ell}/w(t)} - 1 \leq \frac{\lfloor 2^{2\ell}/w(t^*) \rfloor + 1}{\lfloor 2^{2\ell}/w(t) \rfloor} - 1 \\
&\leq \frac{w'(t^*)}{w'(t)} - 1 + \frac{1}{w'(t)} \leq \frac{w'(t^*)}{w'(t)} - 1 + \frac{w'(t^*)}{w'(t)} - 1 \\
&= \alpha\Big(\frac{w'(t^*)}{w'(t)} - 1\Big) = \alpha\Big(\mathrm{R}_{\text{PARTIAL-MVV}}\big(f(x), p\big) - 1\Big).
\end{aligned}$$

2. The only-if-part follows from the completeness of PARTIAL-MVV$_{\exists\mathrm{sol}}$, the if-part from Theorem 4.3.

3. For the completeness of PARTIAL-MVV$_{\exists\mathrm{sol}}$, we reduce GAP to is as follows: map a tuple (G, s, t) to (G, s, w), where $w(t) = 1$ and w is undefined otherwise. □

We can restrict the problem by allowing only small weights, that is, weights taken from the set $\{1, \ldots, n\}$. The resulting problem has the same properties as the unrestricted version, if we replace 'deterministic output' by 'polynomially bounded' everywhere.

PARTIAL-MOST-VALUABLE-VERTEX-POLYNOMIALLY-BOUNDED (PARTIAL-MVV-PB)

Instances. Directed graph $G = (V, E)$, a vertex $s \in V$, and a partial weight function $w\colon V \to \{1, \ldots, |V|\}$.

Solutions. Walk starting at s in G ending at some vertex t for which $w(t)$ is defined.

Measure. Weight $w(t)$ of vertex at which the walk ends.

Theorem 3.2.

1. PARTIAL-MVV-PB *is* $\leq_{\mathrm{E}}^{\log}$*-complete for* $\mathrm{NLO}_{\mathrm{pb}}$.
2. PARTIAL-MVV-PB $\in$ poly-ApxLO *iff* $\mathrm{L} = \mathrm{NL}$.
3. PARTIAL-MVV-PB$_{\exists\mathrm{sol}}$ *and* PARTIAL-MVV-PB$_{\mathrm{opt}>}$ *are* $\leq_{\mathrm{m}}^{\log}$*-complete for* NL.

Proof. 1. The same proof as in Theorem 3.1 works here. The only difference lies in the reduction of the minimisation to the maximisation version. Here, we do not use $2^{2\ell}$ for the 'rescaling', which is too large, but n^2 and increase the graph by $n^2 - n$ many unreachable vertices.

2. The only-if-part follows from the completeness of PARTIAL-MVV-PB$_{\exists\mathrm{sol}}$, the if-part from Theorem 4.3.

3. Once more, the same proof as in Theorem 3.1 applies. □

MOST-VALUABLE-VERTEX (MVV)

Instances. Directed graph $G = (V, E)$, a vertex $s \in V$, and a (total) weight function $w\colon V \to \mathbb{N}^+$.

Solutions. Walk starting at s in G.

Measure. Weight $w(t)$ of vertex t at which the walk ends.

Theorem 3.3.

1. MVV *is* $\leq_{\mathrm{E}}^{\log}$*-complete for* exp-ApxLO$_{\mathrm{do}}$.
2. MVV $\in$ poly-ApxLO *iff* MVV $\in$ LO *iff* L = NL.
3. MVV$_{\exists\mathrm{sol}} \in \mathrm{AC}^0$ *and* MVV$_{\mathrm{opt}>}$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* NL.

Proof. 1. The completeness proof for MVV is similar to the proof of Theorem 3.1. The difference lies in the fact that we must now assign a value to all vertices. In the following, we only consider maximisation problems, since LVV $\leq_{\mathrm{E}}^{\log}$ MVV in the same way as PARTIAL-LVV $\leq_{\mathrm{E}}^{\log}$ PARTIAL-MVV.

Let $P = (I, S, m, \max) \in$ exp-ApxLO$_{\mathrm{do}}$. Let A be a deterministic logspace algorithm that produces solutions for P. Let $m(x, y) \leq 2^{|x|^c}$ for some c. In the following, let us refer to the objects constructed in the proof of Theorem 3.1 by putting a tilde above them. For example, $\tilde{w}$ denotes the weight function constructed in that proof. The new reduction (f, g, α) from P to MVV works as follows: For an instance x, let function f maps x to the triple $(\tilde{G}, \tilde{s}, w)$. The new weight function w is defined differently: For vertices v where $\tilde{w}(v)$ is defined we set $w(v) := 2^{|x|^c}\tilde{w}(v)$. For vertices v where $\tilde{w}(v)$ was undefined, let $w(v) := 1$.

The function g is defined as follows: Let p be a solution for $x := (G, s, w)$ and let t be the last vertex on the walk. If $w(t)$ is undefined (which can be checked in logarithmic space), let $g(x, p) := A(x)$. Otherwise, let $g(x, p) := \tilde{g}(x, p)$. Let $\alpha := 1$.

We must now argue that our construction is ratio preserving. Let $p \in S_P\big(f(x)\big)$ and let t be the last vertex on the walk p. If $\tilde{w}(t)$ is defined, then $m_{\mathrm{MVV}}\big(f(x), p\big) = 2^{|x|^c} m_P\big(x, g(x, p)\big)$. We also have $m^*_{\mathrm{MVV}}\big(f(x)\big) = 2^{|x|^c} m^*\big(x, g(x, p)\big)$. Thus $\mathrm{R}_P\big(x, g(x, p)\big) = \mathrm{R}_{\mathrm{MVV}}\big(f(x), p\big)$.

Now let $\tilde{w}(t)$ be undefined. Then $m_{\mathrm{MVV}}\big(f(x), p\big) = 1$. Since $m_{\mathrm{MVV}}\big(f(x)\big) \geq 2^{|x|^c}$, we conclude $\mathrm{R}_{\mathrm{MVV}}\big(f(x), p\big) \geq 2^{|x|^c}$. In other words, the ratio of the solution p is extremely bad. On the other hand, for all y we have $\mathrm{R}_P(x, y) \leq 2^{|x|^c}$. In other words, no ratio for P can be worse than $2^{|x|^c}$. Thus $\mathrm{R}_P\big(x, g(x, p)\big) \leq \mathrm{R}_{\mathrm{MVV}}\big(f(x), p\big)$.

2. First, let A be a poly-approximator for MVV. We can then solve GAP deterministically in logarithmic space as follows: on input (G, s, t) of total length ℓ, run A on the tuple (G, s, w), where $w(t) = 2^\ell$ and $w(t') = 1$ for $t' \neq t$. For almost all inputs, the poly-approximator A must output a walk to t if there exists one. For the remaining implications, use Theorem 4.3.

3. For the completeness of MVV$_{\mathrm{opt}>}$ we can use the construction for the second claim to reduce GAP to it. We map (G, s, t) to $\big((G, s, w), 2^\ell - 1\big)$. □

MOST-VALUABLE-VERTEX-POLYNOMIALLY-BOUNDED (MVV-PB)

Instances. Directed graph $G = (V, E)$, a vertex $s \in V$, and a weight function $w\colon V \to \{1, \ldots, |V|\}$.

Solutions. Walk starting at s in G.

Measure. Weight $w(t)$ of the vertex t at which the walk ends.

Theorem 3.4.

1. MVV-PB *is* $\leq_{\mathrm{E}}^{\log}$*-complete for* poly-ApxLO$_{\mathrm{pb}}$.
2. MVV-PB $\in$ ApxLO *iff* MVV-PB $\in$ LO *iff* L = NL.
3. MVV-PB$_{\exists\mathrm{sol}} \in \mathrm{AC}^0$ *and* MVV-PB$_{\mathrm{opt}>}$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* NL.

The proof is almost identical to the proof of Theorem 3.3 and hence omitted.

The budget problem of the undirected version of the problem is quite interesting since, for once, it is SL-complete. The budget problems of most other undirected optimisation problems are NL-complete and thus presumably harder to solve.

UNDIRECTED-MOST-VALUABLE-VERTEX (UMVV)

Instances. Undirected graph $G = (V, E)$, a vertex $s \in V$, and a (total) weight function $w\colon V \to \mathbb{N}^+$.

Solutions. Walk starting at s in G.

Measure. Weight $w(t)$ of vertex t at which the walk ends.

Theorem 3.5.

1. UMVV $\in$ exp-ApxLO.
2. UMVV $\in$ poly-ApxLO *iff* UMVV $\in$ LO *iff* L = SL.
3. $\text{UMVV}_{\exists\text{sol}} \in \text{AC}^0$, $\text{UMVV}_{\text{opt}>}$ *is* $\leq_{\text{m}}^{\log}$*-complete for* SL.

Proof. 1. Just output a path containing only s.

2. We can use the same argument as for the second claim of Theorem 3.3 to show that if there exists a poly-approximator for UMVV, then UGAP $\in$ L. Next, assume L = SL. We show UMVV $\in$ LO. Since $\text{UMVV}_{\text{opt}>} \in$ SL, see below, we can compute the most valuable vertex in FL^{SL}. Having found this vertex, we can compute a path to this vertex in FL^{SL} by the first claim of Theorem 3.9.

3. For the completeness of $\text{UMVV}_{\text{opt}>}$, hardness again follows from the argument from the second claim. The tricky part is showing $\text{UMVV}_{\text{opt}>} \in$ SL. Note that this is not as 'obvious' as it may seem: the budget problems of most other undirected optimisation problems are NL-complete, see Theorems 3.9 and 3.20.

We show $\text{UMVV}_{\text{opt}>} \leq_{\text{m}}^{\log}$ UGAP. Let an undirected graph $G = (V, E)$ be given, a start vertex v, a weight function $w\colon V \to \mathbb{N}^+$, and a value $z \in \mathbb{N}^+$. We must construct, in logarithmic space, an undirected graph G' and vertices s' and t' such that there is a path from s' to t' iff there is a vertex v in G reachable from s with $w(v) > z$.

The graph G' is constructed as follows: Let $V' := V \cup \{t'\}$, where t' is a fresh vertex. Let E' contain all edges in E plus an edge between t' and every vertex $v \in V$ with $w(v) > z$. Thus $E' = E \cup \big\{\{t', v\} \mid w(v) > z\big\}$. Let $s' := s$.

To see that this reduction works, first assume that there is a vertex $v \in V$ that is reachable from s for which $w(v) > z$. Then the path to v in G is also a path to v from s' in G'. Since $w(v) > z$ we can go from v to t' in G'. Thus t' is reachable from s' in G'. For the other direction, suppose there exists a path from s' to t' in G'. Let v be the vertex reached by the path directly prior to t'. Then $w(v) > z$ by definition. Furthermore, the path from s' to v in G' is also a path from s to v in G. To see this, note that the path from s' to v cannot use any of the edges we added in G' since the path does not contain the vertex t', while every added edge does. □

MOST-VALUABLE-VERTEX-RATIO-r (MVV-RATIO-r) for rationals $r > 1$.

Instances. Directed graph $G = (V, E)$, a vertex $s \in V$, and a weight function $w\colon V \to \mathbb{N}^+$ with

$$\max\{w(v) \mid v \in V\} \leq r \min\{w(v) \mid v \in V\}.$$

Solutions. Walk starting at s in G.

Measure. The weight of the vertex at which the walk ends.

Theorem 3.6.

1. MVV-RATIO-r *is* $\leq_{\mathrm{E}}^{\log}$*-complete for* $\mathrm{ApxLO}_{\mathrm{do}}$ *for all* $r > 1$.
2. MVV-RATIO-$r \in$ LAS *for any* $r > 1$, *iff* $\mathrm{L} = \mathrm{NL}$.
3. MVV-RATIO-$r_{\exists\mathrm{sol}} \in \mathrm{L}$, MVV-RATIO-$r_{\mathrm{opt}>}$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* NL *for all* $r > 1$.

Proof. 1. The problem can be approximated in logarithmic space with a ratio of r: just output the path containing only s. To prove hardness, we first show that every maximisation problem in $\mathrm{ApxLO}_{\mathrm{do}}$ can be reduced to MVV-RATIO-r for some appropriate $r > 1$. Second, we show that we can reduce MVV-RATIO-r to MVV-RATIO-r' for all $r, r' > 1$. Third, we show LVV-RATIO-$r \leq_{\mathrm{E}}^{\log}$ MVV-RATIO-$2r$ to cover also minimisation problems.

Let $P = (I, S, m, \max) \in \mathrm{ApxLO}_{\mathrm{do}}$. Let A be an r-approximation algorithm for some constant $r > 1$. For the reduction (f, g, α) of P to MVV-RATIO-r, as in the proof of Theorem 3.3 for MVV, let us attach a tilde to denote the objects constructed in the proof of Theorem 3.1 for PARTIAL-MVV.

The instance transformation function f maps x to $(\tilde{G}, \tilde{s}, w)$, where w is defined as follows: Let $w(v) = \tilde{w}(v)$ if $\tilde{w}(v)$ is defined and if $\tilde{w}(v) \geq m\big(x, A(x)\big)$; otherwise let $w(v) = m\big(x, A(x)\big)$. Note that the output of f is a valid instance: the ratio between the maximal value of $w(v)$, which is $m^*(x)$, and the minimal value of $w(v)$, which is $m\big(x, A(x)\big)$, is at most r. The solution transformation function g is defined as follows: Let p be a solution and let t be the last vertex on this walk. If $w(t) > m\big(x, A(x)\big)$, let $g(x, p) := \tilde{g}(x, p)$; otherwise let $g(x, p) := A(x)$. Let $\alpha = 1$. The reduction is approximation preserving since the ratio of any solution $p \in S_{\text{MVV-RATION-}r}(x)$ is exactly the same as the ratio of $g(x, p)$ with respect to P.

Our second aim is to show MVV-RATIO-$r \leq_{\mathrm{E}}^{\log}$ MVV-RATIO-r' for all $r, r' > 1$. We can assume $r < r'$. The reduction (f, g, α) works as follows: The function f takes an instance $x = (G, s, w)$ for MVV-RATIO-r as input and yields the instance (G, s, w') for MVV-RATIO-r' as output, where $w'(v) := w(v) + \beta b$, $b := \max\{w(v) \mid v \in V\}$, and $\beta = \left\lceil \frac{r - r'}{r(r'-1)} \right\rceil$. Note that $\min\{w(v) \mid v \in V\} \geq b/r$. Adding b ensures that $f(x)$ is a valid instance for MVV-RATIO-r':

$$\begin{aligned}\frac{\max\{w'(v) \mid v \in V\}}{\min\{w'(v) \mid v \in V\}} &= \frac{\max\{w(v) \mid v \in V\} + \beta b}{\min\{w(v) \mid v \in V\} + \beta b} = \frac{b(1+\beta)}{\min\{w(v) \mid v \in V\} + \beta b} \\ &\leq \frac{b(1+\beta)}{b(1/r + \beta)} = \frac{r + r\beta}{1 + r\beta} \leq \frac{r + (r - r')/(r' - 1)}{1 + (r - r')/(r' - 1)} = \frac{rr' - r + r - r'}{r' - 1 + r - r'} = r'.\end{aligned}$$

In the calculation, we used $(r + r\lceil z \rceil)/(1 + r\lceil z \rceil) \leq (r + rz)/(1 + rz)$, which holds since $r \geq 1$. The function g maps (x, p) to p. Let $\alpha = 1 + r\beta$. Let p be any solution for $f(x)$ and let t denote the last vertex of p. Let t^* denote the last vertex of a walk in $S^*(x)$. Since $w(t) \geq b/r$, we have

$$\begin{aligned}\mathrm{R}_{\text{MVV-RATIO-}r}\big(x, g(x,p)\big) - 1 = \frac{w(t^*)}{w(t)} - 1 &= \frac{w(t^*) - w(t)}{w(t)} \\ &= \alpha \frac{\big(w(t^*) + \beta b\big) - \big(w(t) + \beta b\big)}{\alpha w(t)} = \alpha \frac{w'(t^*) - w'(t)}{w(t) + \beta r w(t)} \\ &\leq \alpha \frac{w'(t^*) - w'(t)}{w(t) + \beta b} = \alpha\Big(\mathrm{R}_{\text{MVV-RATIO-}r'}\big(f(x), p\big) - 1\Big).\end{aligned}$$

Our third and final aim is to show LVV-RATIO-$r \leq_{\mathrm{E}}^{\log}$ MVV-RATIO-$2r$. We use exactly the same reduction as the one used to show PARTIAL-LVV $\leq_{\mathrm{E}}^{\log}$ PARTIAL-MVV. We only have to argue that this reduction really maps instances for LVV-RATIO-r to instances for MVV-RATIO-$2r$. This can be

seen as follows:

$$\frac{\max\{\lfloor 2^{2\ell} w(v)\rfloor \mid v \in V\}}{\min\{\lfloor 2^{2\ell} w(v)\rfloor \mid v \in V\}} \leq \frac{\max\{2^{2\ell} w(v) \mid v \in V\}}{\min\{2^{2\ell} w(v) \mid v \in V\} - 1} \leq \frac{\max\{2^{2\ell} w(v) \mid v \in V\}}{\min\{2^{2\ell} w(v) \mid v \in V\}/2}$$
$$= 2\frac{\max\{w(v) \mid v \in V\}}{\min\{w(v) \mid v \in V\}} = 2r.$$

2. For the only-if-part, suppose MVV-RATIO-$r \in$ LAS for some $r = p/q > 1$, where p and q are positive integers. Let A be the logspace approximation scheme. We construct a deterministic logspace decision algorithm for GAP. On input (G, s, t), we construct a tuple (G, s, w) with $w(t) = p$ and $w(t') = q$ for $t \neq t'$. This tuple is clearly an instance for MVV-RATIO-r. We can then run A on this instance for the ratio $r - (r-1)/2$. We accept if the measure of A's output is p, otherwise we reject. To see that this algorithm works, note that if t is reachable from s, then A must output a walk to t, since any other walk has measure q and ratio $p/q = r > r - (r-1)/2$. The if-part follows from Theorem 4.3.

3. For the budget problem, we can use the construction from the second claim in order to reduce GAP to MVV-RATIO-$r_{\text{opt}>}$. On input (G, s, t) we ask whether $\big((G, s, w), p\big)$ is in MVV-RATIO-$r_{\text{opt}>}$. □

The last variant of the most valuable vertex problem is rather artificial, but it is useful for separating LAS from FLAS under appropriate assumptions. Recall that $\log^* n$ is the iterated logarithm and that the tower function tow is its inverse.

MOST-VALUABLE-VERTEX-RATIO-DECREASING (MVV-RATIO-DEC).

Instances. Directed graph $G = (V, E)$, a vertex $s \in V$, and a weight function $w\colon V \to \mathbb{N}^+$ with

$$\max\{w(v) \mid v \in V\} \leq \left(1 + \frac{1}{\log^* |V|}\right) \min\{w(v) \mid v \in V\}.$$

Solutions. Walk starting at s in G.

Measure. The weight of the vertex at which the walk ends.

Theorem 3.7.

1. MVV-RATIO-DEC $\in$ LAS.
2. MVV-RATIO-DEC $\notin$ FLAS, *unless* NL $\subseteq$ DSPACE$[\log^* n \log n]$.
3. MVV-RATIO-DEC$_{\exists\text{sol}} \in$ L, MVV-RATIO-DEC$_{\text{opt}>}$ *is* $\leq_{\text{m}}^{\log}$*-complete for* NL.

Proof. 1. We can check in logarithmic space, whether a tuple (G, s, w) is an instance. The logspace approximation scheme for MVV-RATIO-DEC works as follows: On input of an instance (G, s, w) and a desired ratio $r = 1 + 1/k$, we check whether $1 + 1/\log^* |V| \leq r$. If so, we output the path containing just the first vertex. If not, that is, if the graph is not large enough such that the trivial solution automatically satisfies the ratio, we brute force the optimal solution. This search needs space $O\big(\text{tow}\, k \log n\big)$.

2. Suppose MVV-RATIO-DEC $\in$ FLAS via an approximation scheme A with space requirements $O\big(\log k \log |x|\big)$. We argue that we can decide GAP in space $O(\log^* n \log n)$. Let (G, s, t) be given with $G = (V, E)$ and $n = |V|$. Construct an instance (G, s, w) for MVV-RATIO-DEC by setting $w(t) = 1 + \log^* n$ and $w(t') = \log^* n$ for $t' \neq t$. Run A on (G, s, w) and $k = 2\log^* n$, that is, for a desired ratio of $1 + 1/(2\log^* n)$. We accept iff the walk produced by A ends at t. Note that if there is a path from s to t in G, the algorithm A cannot output a walk that does not end at t

since any such walk has a ratio of $1 + 1/\log^* n > 1 + 1/(2\log^* n)$. The space needed by A is $O(\log k \log n) = O(\log\log^* n \log n) \subseteq O(\log^* n \log n)$.

3. For the budget problem, we reduce GAP to MVV-RATIO-DEC$_{\text{opt}>}$ as follows: map (G, s, t) to $\big((G, s, w), \log^* n\big)$ with $w(t) = 1 + \log^* n$ and $w(t') = \log^* n$ for $t' \neq t$. □

3.2 Shortest Path Problems

The shortest path problem we study is to find the shortest path in a directed graph. Both the budget problem and the existence problems underlying the shortest path problem for undirected graphs are canonical NL-complete problems. For other types of graphs, the situation is more complex. In this section, we treat the shortest path problems for undirected graphs, directed graphs, forests, tournaments, graphs with bounded independence number, and undirected planar-embedded graphs.

We start with directed graphs. The first of the claims of the next theorem is a 'constructive version' of Savitch's theorem [18]. It tells us that we can find the shortest path in space $O(\log \ell \log n)$, where ℓ is the length of this path.

SHORTEST-PATH (SP)
Instances. Directed graph $G = (V, E)$, two vertices $s, t \in V$.
Solutions. Walk from s to t in G.
Measure. Length of the walk.

Theorem 3.8.

1. *Optimal solutions for* SP *can be found in space* $O(\log \ell \log |V|)$*, where* ℓ *is the length of the shortest path, or* $O(\log^2 |V|)$ *if no path exists.*
2. SP *is* $\leq_{\text{E}}^{\log}$*-complete for* NLO_{pb}.
3. SP $\in$ exp-ApxLO *iff* SP $\in$ LO *iff* L = NL.
4. $\text{SP}_{\exists\text{sol}}$ *and* $\text{SP}_{\text{opt}<}$ *are* $\leq_{\text{m}}^{\log}$*-complete for* NL.

Proof. 1. The rough idea is the same as Savitch's. The main difference is that we do not only need to check whether there exists a path between a source s and a target t in a directed graph $G = (V, E)$, but we also have to output such a path. If there are several paths, we must 'decide on one of them' and we must do so 'within the recursion'.

We use two procedures. The first, called reachable(u, v, ℓ), checks whether there *exists* a path from u to v of length ℓ. This procedure is the standard procedure from Savitch's theorem. The second procedure is called construct(u, v, ℓ). It outputs a path from u to v of length ℓ, provided such a path exists. If no such path exists, it may output nonsense (and should thus not be called).

The procedure reachable(u, v, ℓ) is defined as follows:

procedure reachable(u, v, ℓ)
  let *flag* := false
  if $\ell = 0 \wedge u = v$ **then** *flag* := true
  if $\ell = 1 \wedge (u, v) \in E$ **then** *flag* := true
  if $\ell > 1$ **then**
    forall $z \in V$ **do**
      if reachable$\big(u, z, \lfloor \ell/2 \rfloor\big) \wedge$ reachable$\big(z, v, \ell - \lfloor \ell/2 \rfloor\big)$ **then** *flag* := true
  return *flag*

The algorithm needs space $O(\log \ell \log n)$. Building on the above procedure, we can now construct an output algorithm. On input (u, v, ℓ) the algorithm will output a path from u to v of

length ℓ. However, the output of the algorithm will not include the vertex v itself. This makes it easier to assemble the outputs from recursive calls to a single path.

procedure construct(u, v, ℓ)
  if $\ell \leq 1$ **then output** u
  else
    forall $z \in V$ **do**
      if reachable$\big(u, z, \lfloor \ell/2 \rfloor\big) \wedge$ reachable$\big(z, v, \ell - \lfloor \ell/2 \rfloor\big)$ **then**
        call construct$\big(u, z, \lfloor \ell/2 \rfloor\big)$
        call construct$\big(z, v, \ell - \lfloor \ell/2 \rfloor\big)$
        exit

This algorithm also needs space $O(\log \ell \log n)$. The complete algorithm that produces optimal solutions for SP works as follows: For each $\ell \in \{1, \ldots, |V|\}$ the algorithm calls reachable$\big(s, t, \ell\big)$. For the smallest ℓ for which this test succeeds, it calls construct$\big(s, t, \ell\big)$, appends t to the output, and quits.

2. Clearly SP $\in$ $\mathrm{NLO_{pb}}$. To prove completeness, we show PARTIAL-LVV-PB $\leq_{\mathrm{E}}^{\log}$ SP via a reduction (f, g, α). The problem PARTIAL-LVV-PB is $\leq_{\mathrm{E}}^{\log}$-complete for $\mathrm{NLO_{pb}}$ by Theorem 3.2.

The instance transformation function f gets an instance (G, s, w) for PARTIAL-LVV-PB as input. Let $G = (V, E)$ and let $n := |V|$. The function f maps this instance to an instance (G', s', t') for SP as follows: The graph G' is obtained from G by adding a new vertex t and adding a new directed line graph from v to t of length exactly $w(v)2n$ for each vertex $v \in V$ for which $w(v)$ is defined. Set $s' := s$. Let us call the vertices added during the construction of G' *added vertices* and let us call the vertices copied from G *original vertices*.

The solution transformation function g maps a walk p from s' to t' to the initial segment of p that contains the original vertices. Let $\alpha := 4$.

We have to show that this reduction is error preserving. To see this, let $x = (G, s, w)$ be an instance and let p be a walk from s' to t'. Let v denote the last original vertex on p. Let v^* denote a vertex in G that is reachable from s whose weight $w(v^*)$ is defined and minimal. If $w(v) = w(v^*)$, then $\mathrm{R}_{\text{PARTIAL-LVV-PB}}\big(x, g(x, y)\big) = 1$ and we are done. Otherwise $w(v) - w(v^*) \geq 1$ and hence $w(v)2n - w(v^*)2n - 2n \geq 0$. The walk p has length at least $w(v)2n$. The shortest path from s' to t' has length at most $w(v^*)2n + n$. With these observation, we can bound the ratios as follows:

$$\begin{aligned}
\mathrm{R}_{\text{PARTIAL-LVV-PB}}\big(x, g(x, y)\big) - 1 &= \frac{w(v) - w(v^*)}{w(v^*)} = \frac{w(v)2n - w(v^*)2n}{w(v^*)2n} \leq 2\frac{w(v)2n - w(v^*)2n}{w(v^*)2n + n} \\
&\leq 2\frac{w(v)2n - w(v^*)2n + w(v)2n - w(v^*)2n - 2n}{w(v^*)2n + n} \\
&= 4\frac{w(v)2n - w(v^*)2n - n}{w(v^*)2n + n} \\
&= 4\Big(\frac{w(v)2n}{w(v^*)2n + n} - 1\Big) \leq \alpha\Big(\mathrm{R}_{\text{SP}}\big(f(x), y\big) - 1\Big).
\end{aligned}$$

3. We establish a cyclic implication between the three statements. If SP $\in$ exp-ApxLO, then we have $\mathrm{SP}_{\exists\text{sol}} \in$ L by Theorem 2.12 and hence L = NL. If L = NL, then by Theorem 4.3 we get $\mathrm{NLO_{do}} \subseteq$ LO.

4. The decision problem $\mathrm{SP}_{\exists\text{sol}}$ is exactly GAP, which is well-known to be NL-complete. □

The next optimisation problem asks us to find the shortest path in an undirected graph. Its existence problem is well-studied. It is complete for the class SL of symmetric logspace by definition.

Surprisingly, its budget problem is still NL-complete. Thus it is presumably easier to find an arbitrary path in an undirected graph than finding the shortest one. Furthermore, the shortest path problem for directed graphs can be $\leq^{\log}_{\text{1-tt}}$-reduced to the shortest path problem for undirected graphs.

UNDIRECTED-SHORTEST-PATH (USP)
Instances. Undirected graph $G = (V, E)$, two vertices $s, t \in V$.
Solutions. Walk from s to t in G.
Measure. Length of the walk.

Theorem 3.9.

1. *There is a function in* $\mathrm{FL}^{\mathrm{SL}}$ *that produces solutions for* USP.
2. USP *is* $\leq^{\log}_{\text{1-tt}}$*-complete for* $\mathrm{NLO}_{\mathrm{pb}}$.
3. $\mathrm{USP} \in \text{exp-ApxLO}$ *iff* $\mathrm{L} = \mathrm{SL}$.
4. $\mathrm{USP} \in \mathrm{LO}$ *iff* $\mathrm{L} = \mathrm{NL}$.
5. $\mathrm{USP}_{\exists\mathrm{sol}}$ *is* $\leq^{\log}_{\mathrm{m}}$*-complete for* SL, $\mathrm{USP}_{\mathrm{opt}<}$ *is* $\leq^{\log}_{\mathrm{m}}$*-complete for* NL.

For the proof of the theorem, we need two lemmata. The second lemma is an extension of a result due to Nisan and Ta-Shma [15], who prove SL = coSL.

Lemma 3.10. *There exists a function in* FL *that maps a directed graph G, two vertices s and t, and a number ℓ (coded in unary) to a connected, undirected graph G' and two vertices s' and t', such that* $\mathrm{d}_G(s,t) \leq \ell$ *iff* $\mathrm{d}_{G'}(s',t') \leq \ell$.

Proof. Let G, s, t, and ℓ be given. Let $G = (V, E)$ and let $n := |V|$. The undirected graph $G' = (V', E')$ is produced as follows: The vertex set V' is given by $\{1, \ldots, \ell+1\} \times V$. We can think of this vertex set as a big grid consisting of $\ell + 1$ rows and n columns. There is an (undirected) edge in G' between a vertex (r_1, v_1) and (r_2, v_2) iff

1. $r_2 = r_1 + 1$ and $(v_1, v_2) \in E$, that is, vertices on neighbouring rows are connected if they are connected in G, or
2. $r_2 = r_1 + 1$ and $v_1 = v_2$, that is, the same vertices are connected on neighbouring rows, or
3. $r_2 = r_1$, that is, all vertices on the same row are connected.

Let $s' := (1, s)$ and $t' := (\ell + 1, t)$. This graph is clearly connected.

To see that G' has the desired properties, first assume that there exists a path from s to t in G of length $m \leq \ell$. Let $(v_1, \ldots, v_m)$ with $v_1 = s$ and $v_m = t$ be this path. Then $(1, v_1)$, $(2, v_2)$, $\ldots$, (m, v_m), $(m + 1, v_m)$, $\ldots$, (ℓ, v_m) is a path in G' of length at most ℓ. Second, assume that there exists a path from s' to t' in G' of length $m \leq \ell$. Then $m = \ell$, since any path from the first row to the last row must 'brave all rows'—there are no edges that allow us to skip a row. Let $(v'_1, \ldots, v'_\ell)$ be this path. Then $v'_i = (i, v_i)$ for some vertices $v_i \in V$. The sequence $(v_1, \ldots, v_\ell)$ is 'almost' a path from s to t in G: For any $i \in \{1, \ldots, \ell - 1\}$ we either have $v_i = v_{i+1}$ or $(v_i, v_{i+1}) \in E$. Thus, by removing consecutive duplicates, we obtain a path from s to t. □

Lemma 3.11 (Spanning Forest Lemma). *Let $A \in \mathrm{L}$ be a class of (the codes of) undirected graphs that is closed under edge deletion. Let A-*USP *denote the undirected shortest path problem whose instance set is restricted to graphs in A. Then there is a function in* $\mathrm{FL}^{A\text{-}\mathrm{USP}_{\exists\mathrm{sol}}}$ *that produces solutions for A-*USP.

Proof. We first show that there exists a function $f \in \mathrm{FL}^{A\text{-}\mathrm{USP}_{\exists\mathrm{sol}}}$ that maps an instance $(G, s, t) \in I$ to a spanning forest of G. A *spanning forest* of a graph $G = (V, E)$ is a forest $G' = (V, E')$ with $E' \subseteq E$ such that there is a path between any two vertices $u, v \in V$ in G iff there is a path between them in G'. Once we have constructed such a spanning forest, we can decide whether there is a path from s to t in G by checking whether there is a path from s to t in G'. Provided there is a path between s and t, by Theorem 3.14 we can construct such path in the forest G' in logarithmic space. This path is also a path in G.

Let (G, s, t) be given, let $G = (V, E) \in A$, and let $n := |V|$. For the following argument, it will be useful to think of the elements of E as sets $\{u, v\}$ rather than as tuples. Let $E = \{e_1, \ldots, e_m\}$. Let $G_i = \big(V, \{e_1, \ldots, e_i\}\big)$. Note that $G_i \in A$ for all $i \in \{1, \ldots, m\}$. We obtain E' from E as follows: we remove all edges $e_i = \{u, v\}$ for which there exists a path from u to v in G_{i-1}. Clearly, E' can be constructed in logarithmic space with oracle access to $A\text{-}\mathrm{USP}_{\exists\mathrm{sol}}$.

We must now argue that G' is a spanning forest of G. First, we argue by induction on i that for any vertex u the set of vertices reachable from u in G_i is the same as the set of vertices reachable from u in $G'_i := \big(V, E' \cap \{e_1, \ldots, e_i\}\big)$. For $i = 0$ this is obviously correct. Suppose the claim has been shown for some i. Since G_i and G_{i+1} differ in just one edge, namely e_{i+1}, there can be at most one new vertex $v \in V$ that is reachable from u in G_{i+1}, but not in G_i. This vertex must be connected, by e_{i+1}, to some vertex v' in G_i that is reachable from u. By assumption, v' is reachable from u in G'_i. If $e_{i+1} \in E'$, then v is reachable from u in G'_{i+1} and we are done. Otherwise, by construction, v is reachable from u in G_i, which is a contradiction.

Second, we must show that G' is a forest. Suppose the exists a cycle in G'. Let i be the highest index such that $e_i = \{u, v\}$ is an element of this cycle. Then there is a path from u to v in G_{i-1} and thus $e_i \notin E'$, a contradiction. □

Proof of Theorem 3.9. 1. The first claim follows from Lemma 3.11, setting A to be the class of all undirected graphs. Note that, although we can even construct the shortest path in the spanning forest, this shortest path is not necessarily the shortest path in G.

2. Clearly $\mathrm{USP} \in \mathrm{NLO}_{\mathrm{pb}}$. To prove completeness, we show $\mathrm{SP} \leq^{\log}_{\text{1-tt}} \mathrm{USP}$ via a machine M. We start with a directed graph $G = (V, E)$ and two vertices $s, t \in V$ as input. Let $n := |V|$. For each $\ell \in \{1, \ldots, n\}$ we invoke Lemma 3.10 to obtain a connected, undirected graph G'_ℓ. Our single query q to USP is obtained by writing the graphs G'_1, … , G'_n alongside each other and by connecting the target vertex of each G'_i with the source vertex of the next graph G'_{i+1}. The global source vertex is the source vertex of G'_1, the global target is the target vertex of G'_n.

Consider an answer to this query. It is a path p of minimal length from the global source to the global target. Note that such a path must exist, since all G'_i are connected. The path must pass through all G'_i. Inside each G'_i the path induces a shortest path p_i from s'_i to t'_i.

We find the shortest path from s to t in G by running the following algorithm on the path p: Find the smallest i such that the path p_i inside G'_i has length i. Then, by Lemma 3.10, the shortest path from s to t in G has length i. This path is given by p_i, with the first components removed.

3. This follows from the first and the fifth claim.

4. Assume $\mathrm{USP} \in \mathrm{LO}$. Then $\mathrm{USP}_{\mathrm{opt}<} \in \mathrm{LO}$ by Theorem 2.7. By the fourth claim this implies $\mathrm{L} = \mathrm{NL}$. The other direction follows from Theorem 4.3.

5. We have $\mathrm{USP}_{\exists\mathrm{sol}} = \mathrm{UGAP}$, which is SL-complete by definition. To prove the NL-completeness of $\mathrm{USP}_{\mathrm{opt}<}$, we reduce GAP to it. On input of a directed graph G, we apply the construction of Lemma 3.10 to this graph for $\ell = n$ and obtain a triple (G', s', t'). We then ask 'is there a path from s' to t' in G' of length at most n?'. By Lemma 3.10, the answer to this question will be 'yes' iff there is a path from s to t in G (of length at most n). □

We now study an optimisation problem that has a logspace optimisation scheme, but whose budget problem is still NL-complete. This problem asks us to find the shortest path between two vertices in a tournament. A tournament is a directed graph in which there is exactly one edge between any two different vertices.

TOURNAMENT-SHORTEST-PATH (TOURNAMENT-SP)
Instances. Tournament graph $G = (V, E)$, two vertices $s, t \in V$.
Solutions. Walk from s to t in G.
Measure. Length of the walk.

Theorem 3.12.

1. TOURNAMENT-SP *is* $\leq^{\log}_{1\text{-tt}}$*-complete for* $\mathrm{NLO_{pb}}$.
2. TOURNAMENT-SP *is* $\leq^{\log}_{\mathrm{E}}$*-complete for* $\mathrm{NLO_{pb}}$ *iff* $\mathrm{L} = \mathrm{NL}$.
3. TOURNAMENT-SP $\in$ FLAS.
4. TOURNAMENT-SP $\in$ LO *iff* $\mathrm{L} = \mathrm{NL}$.
5. $\text{TOURNAMENT-SP}_{\exists\mathrm{sol}} \in \mathrm{AC}^0$, $\text{TOURNAMENT-SP}_{\mathrm{opt}<}$ *is* $\leq^{\log}_{\mathrm{m}}$*-complete for* NL.

Once more, we first need a lemma.

Lemma 3.13. *There exists a function in* FL *that maps a directed graph G, two vertices s and t, and a number ℓ (coded in unary) to a strongly connected tournament G' and two vertices s' and t' such that* $\mathrm{d}_G(s,t) \leq \ell$ *iff* $\mathrm{d}_{G'}(s',t') \leq \ell$.

Proof. The proof is similar to the proof of Lemma 3.10. We use the same vertex set V', but the edge relation is defined differently, because we have to construct a tournament. Let $<$ be a linear ordering of V' such that s is minimal and t is maximal. There is an edge in G' from a vertex (r_1, v_1) to a vertex (r_2, v_2) iff any of one of the following condition holds:

1. $r_2 = r_1 + 1$ and $(v_1, v_2) \in E \cup \{(v, v) \mid v \in V\}$. That is, if v_1 and v_2 are connected in G or if $v_1 = v_2$, then there is an edge leading 'downwards' between them on adjacent rows.
2. $r_1 = r_2$ and $v_1 < v_2$, except for $(v_1, v_2) = (s, t)$, where we put in an edge from s to t. That is, the vertices on the same row form a big 'ring'.
3. $r_2 = r_1 - 1$ and $(v_1, v_2) \notin E \cup \{(v, v) \mid v \in V\}$. That is, if v_1 and v_2 are not connected in G and if they are not identical, then there is an edge leading 'upwards' between them on adjacent rows.
4. $r_2 \leq r_1 - 2$. That is, all edges spanning at least two rows point 'upwards'.

The constructed tournament is strongly connected and it has the desired properties. This can be seen by copying the rest of the proof of Lemma 3.10 in verbatim. □

Proof of Theorem 3.12. 1. We can argue similarly to the proof of Theorem 3.9. We only have to ensure that the graph obtained by connecting all the tournaments G'_i must be a tournament once more. This is ensured by directing all missing edges from vertices in G'_i to G'_j whenever $j < i$. The resulting tournament has the same properties as the graph obtained in Theorem 3.9 and we can apply the same algorithm to it, in order to obtain the shortest path from s to t in G.

2. For the if-part, $\mathrm{L} = \mathrm{NL}$ implies $\mathrm{LO_{pb}} = \mathrm{NLO_{pb}}$ by Theorem 4.3 and all problems in $\mathrm{LO_{pb}}$ are complete for it. For the only-if part, suppose TOURNAMENT-SP is $\leq^{\log}_{\mathrm{E}}$-complete for $\mathrm{LO_{pb}}$. Then SP $\leq^{\log}_{\mathrm{E}}$ TOURNAMENT-SP via some reduction (f, g, α). We show GAP $\in$ L. Let a graph (G, s, t) be given. We consider this graph as an instance for SP and apply f to it, obtaining a triple (G', s', t')

in which G' is a tournament. By the third claim, we can compute a path from s' to t' in logarithmic space. If no path exists, we reject (G, s, t). If a path exists, we apply g to it, obtaining a path in G. If this path leads from s to t, we accept; otherwise we reject.

3. For the logspace approximation scheme, let a problem instance (G, s, t) be given and a ratio $r = 1 + 1/k$. Let $d := \mathrm{d}(s, t)$ denote the distance between s and t. Our aim is to construct a path from s to t in G of length at most $d + d/k$. The construction should use space at most $O(\log k \log n)$. Since we can check in logarithmic space whether there exists *any* path from s to t, see below, in the following we may assume that t is reachable from s.

The algorithm consists of a loop that uses only space $O(\log k \log n)$. At the beginning of each iteration we have a tape initialised with a *current vertex* v, which is guaranteed to be reachable from s. During each iteration we find a vertex v', using a procedure described later, that has the following two properties:

1. $\mathrm{d}(v', t) \leq \mathrm{d}(v, t) - 2k$. That is, v' is at least $2k$ steps nearer to t.
2. The distance $\mathrm{d}(v, v')$ is at most $2k + 2$.

Having found v', we output the minimal path from v to v' and replace the current vertex v by v'. By Theorem 3.8, we can output the shortest path from v to v' using only $O(\log k \log n)$ space. We end the loop, when the distance between the current vertex and t is less than $2k + 2$. Let ℓ be the number of iterations we perform, where the last step, in which we directly go from v to t, counts as another iteration. Then the length of the path we output is at most $(2k + 2)\ell$. Since in each step we reduce the distance between the current vertex and t by at least $2k$, we have $\ell \leq d/2k$. Thus the length of the path we output is at most $(2k + 2)\ell \leq (2k + 2)d/2k = d + d/k$.

It remains to show how we can obtain v'. We pick v' according to the following rule: Let S denote the set of vertices reachable from v in exactly $2k + 2$ steps and let v' be a king of S. A king of a graph is a vertex such that all other vertices of the graph are reachable in at most two steps from it. It is well-known, see for example [12], that every tournament has a king. Using the procedure reachable$(v, u, 2k + 2)$ from Theorem 3.8, we can obtain v' in space $O(\log k \log n)$.

The thusly obtained vertex v' has the two desired properties. Trivially, it satisfies the second condition. For the first condition, note that at least one vertex in S has distance $d(v, t) - 2k - 2$, namely the vertex on the shortest path from v to t. However, all vertices in S have distance at most 2 from the king v'. Thus v' has distance at most $d(v, t) - 2k$ from t.

4. The only-if-part follows from the completeness of TOURNAMENT-SP$_{\mathrm{opt}<}$, the if-part from Theorem 4.3.

5. For the complexity of the existence problem see [13, 19]. For the budget problem, we argue exactly the same way as in Theorem 3.9, only invoking Lemma 3.13 instead of Lemma 3.10. □

The previous problems may have created the impression that the budget problem of any reachability problem is NL-complete. However, the problem of finding the shortest path in a *forest* is in LO.

FOREST-UNDIRECTED-SHORTEST-PATH (FOREST-USP)

Instances. Undirected forest $G = (V, E)$ with $\alpha(G) \leq k$, two vertices $s, t \in V$.

Solutions. Walk from s to t in G.

Measure. Length of the walk.

Theorem 3.14.

1. FOREST-USP $\in$ LO.

2. FOREST-USP$_{\exists\mathrm{sol}}$ *and* FOREST-USP$_{\mathrm{opt}<}$ *are* $\leq_{\mathrm{m}}^{\mathrm{AC}^0}$*-complete for* L.

Proof. 1. An undirected tree is a forest iff it is cycle-free, which is known to be an L-complete problem. We can check whether there is *any* path from s to t by starting a walk at s and always 'turning right' at each vertex. If the walk does not hit t after $2|V|$ many steps, there does not exist a path from s to t. In order to compute the *shortest* path from s to t we proceed as follows: First, we output s and make s our *current vertex*. Then we repeatedly apply the following procedure to the current vertex: for each of its neighbours, we check whether t is reachable from this neighbour in the graph obtained by removing the edge connecting the current vertex and the neighbour. There is exactly one vertex for which this test succeeds. We output this vertex, make it the new current vertex, and repeat the procedure until we hit t.

2. For the completeness results note that the symmetric closure of the configuration graph of a logspace machine is a forest. □

Graphs with a bounded independence number are generalisations of tournaments. The *independence number* $\alpha(G)$ of a graph G is the maximum number of vertices that can be picked from G such that no two edges are connected. Tournaments have independence number 1.

INDEPENDENCE-α-SHORTEST-PATH (IND-α-SP) for positive integers α

Instances. Directed graph $G = (V, E)$ with $\alpha(G) \leq \alpha$, two vertices $s, t \in V$.

Solutions. Walk from s to t in G.

Measure. Length of the walk.

Theorem 3.15.

1. IND-α-SP *is* $\leq^{\log}_{\text{1-tt}}$*-complete for* NLO_{pb},
2. IND-α-SP $\in$ FLAS *for all* α, IND-α-USP $\in$ LO *for all* α.
3. IND-α-SP $\in$ LO *for any* α, *iff* L = NL.
4. IND-α-$\text{SP}_{\exists\text{sol}} \in \text{AC}^0$, IND-$\alpha$-$\text{SP}_{\text{opt}<}$ *is* $\leq^{\log}_{\text{m}}$*-complete for* NL *for all* α.

Proof. 1. This follows from the $\leq^{\log}_{\text{1-tt}}$-completeness of TOURNAMENT-SP.

2. For the logspace approximation scheme for IND-α-SP, similarly to the proof for tournaments, see Theorem 3.12, let a graph $G = (V, E)$, vertices $s, t \in V$, and a ratio $r = 1 + 1/k$ be given. Roughly spoken, the algorithm works as in Theorem 3.12, only this time we do not have a single current vertex, but a set C of up to α many. For a set C let $\text{d}_G(C, t) := \min\{\text{d}_G(v, t) \mid v \in C\}$. Let $n := |V|$ and let $d := \text{d}_G(s, t)$ denote the distance between s and t.

The path construction algorithm is a loop in which we construct a sequence $C_1, C_2, \ldots, C_\ell \subseteq V$ of vertex sets with $C_1 = \{s\}$ and $C_\ell = \{t\}$. For the construction of C_{i+1} we access only C_i and use space $O(\log k \log n)$. Once we have constructed C_{i+1} we erase C_i and reuse the space it occupied. The sets C_i have the following properties for $i \in \{2, \ldots, \ell - 1\}$:

1. All elements of C_i are reachable from s.
2. $|C_i| \leq \alpha$.
3. $\text{d}_G(C_i, t) \leq \text{d}_G(C_{i-1}, t) - 2k$ and hence $\text{d}_G(C_i, t) \leq d - 2k(i - 1)$.
4. $\text{d}_G(C_{i-1}, v) = 2k + 2$ for all $v \in C_i$.

For $i = \ell$, the first two conditions also hold and the last one becomes $\text{d}_G(C_{i-1}, t) \leq 2k + 2$. In each iteration we reduce the distance between C_i and t by at least $2k$. Thus $\ell - 1 \leq d/2k$.

The set C_i is obtained from C_{i-1} as follows: If $\text{d}_G(C_{i-1}, t) \leq 2k + 1$, let $C_i := \{t\}$. Otherwise let $S := \{v \in V \mid \text{d}_G(C_{i-1}, v) = 2k + 2\}$ and choose C_i as a 2-dominating set of S of size α. Since $\alpha(G) \leq \alpha$, such a set exists as shown in [13]. We can obtain such a set in space $O(\log k \log n)$ since the question '$v \in S$?' can be answered in space $O(\log k \log n)$ by Theorem 3.8.

In order to output the desired path from s to t of length at most $d + d/k$ we first construct a forest that contains this path. The forest is not actually written down anywhere (we are allowed only a logarithmic amount of space). Rather, as in the proof of FL being closed under composition, the forest's code is dynamically recalculated in space $O(\log k \log n)$ whenever one of its bits is needed. Finding the shortest path in a forest can be done in logarithmic space by Theorem 3.14, and the shortest path in the forest will be the desired path.

The forest is defined as follows: For each $i \in \{2, \dots, \ell\}$ define a 'small' forest F_i as follows: For each $v \in C_i$ it contains the vertices and edges of the shortest path from C_{i-1} to v. This path is constructed by calling the procedure construct(c, v, e) from Theorem 3.8 for the first vertex $c \in C_{i-1}$ for which $e := \mathrm{d}_G(c, v)$ is minimal. Since $\mathrm{d}_G(c, v) \leq 2k+2$, this call needs space $O(\log k \log n)$. The graph F_i is, indeed, a forest since if two paths output by construct for the same source vertex split at some point, they split permanently. The forest F is the union of all the forests F_i constructed during the run of the algorithm.

Consider the shortest path in the forest F. This path passes all C_i. For $i \in \{1, \dots, \ell\}$ let $c_i \in C_i$ be the last vertex of C_i this path visits. Then the total length of the path is given by $\sum_{i=1}^{\ell-1} \mathrm{d}_G(c_i, c_{i+1})$. We have $\mathrm{d}_G(c_i, c_{i+1}) = 2k + 2$ for $i \in \{1, \dots, \ell - 2\}$. Thus the total length is

$$\begin{aligned}(2k+2)(\ell-2) + \mathrm{d}_G(c_{\ell-1}, t) &= (2k+2)(\ell-2) + \mathrm{d}_G(C_{\ell-1}, t) \\ &\leq (2k+2)(\ell-2) + d - 2k(\ell-2) \\ &= d + 2(\ell-2) \leq d + 2(\ell-1) \\ &\leq d + d/k.\end{aligned}$$

For the first inequality we used the third property of the sets C_i, for the last inequality we used the inequality $\ell - 1 \leq d/2k$.

For the undirected version, note that in an undirected graph G with $\alpha(G) \leq \alpha$ there is a path between any two vertices s and t iff there is a path between them of length at most 2α.

3. The only-if-part follows from the completeness of IND-1-SP$_{\mathrm{opt}<}$, the if-part from Theorem 4.3.

4. For the budget problem, note that TOURNAMENT-SP$_{\mathrm{opt}<} \leq_{\mathrm{m}}^{\log}$ IND-1-SP$_{\mathrm{opt}<}$. For the existence problem see [13]. □

The next graph type we study are planar-embedded graphs, which arise in numerous applications. A planar graph 'can be painted on a sheet of paper without edges crossing each other'. A planer-embedded graph is a graph that 'is painted on a sheet of paper without edges crossing each other'. That is, for planar-embedded graphs we are already given the embedding, whereas for planar graphs it is only known that such an embedding exists.

In the following, I show that the reachability problem for undirected planar-embedded graphs is logspace poly-approximable. Thus there exists a logspace algorithm that on input of an undirect graph together with a planar embedding outputs a path from the source to the target or outputs that no such path exists. In particular, this shows that the reachability problem for undirect planar-embedded graphs in in L. Most intriguingly, it is not clear at all, whether the problem in L-complete *despite the fact the reachability for undirected forests is* L*-complete*. Note that undirected forests are obviously planar. The problem is that we do not know how to compute an embedding efficiently.

Before we proceed, let us first formalise the notion of a planar-embedded graph. A *planar-embedded* graph is an edge-labelled graph $G = (V, E)$ such that the following conditions are met:

1. $V \subseteq \mathbb{N} \times \mathbb{N}$,

2. to each edge $(v, v') \in E$ the edges-labelling function η assigns a polygon traverse from v to v' with corners in $\mathbb{N} \times \mathbb{N}$,
3. for any two edges $(v_1, v_1'), (v_2, v_2') \in E$ the polygon traverses $\eta(v_1, v_1')$ and $\eta(v_2, v_2')$ are disjoint in the real plane, except that end points may coincide.

A graph is *planar* if it isomorphic to a planar-embedded graph.

PLANAR-EMBEDDED-UNDIRECTED-SHORTEST-CYCLE (PE-USP)

Instances. Planar-embedded graph $G = (V, E, \eta)$, two vertices $s, t \in V$.

Solutions. Walk from s to t in G.

Measure. Length of the walk (length of walk sequence, not Euclidean distance).

Theorem 3.16.

1. PE-USP *is in* poly-ApxLO$_{\text{pb}}$.

2. PE-USP$_{\exists\text{sol}} \in$ L.

Proof. 1. This claim follows from the second claim and Lemma 3.11 instantiated for the class A of undirected planar-embedded graphs.

2. In this proof, it will sometimes be useful to consider undirected edge between vertices v and v' as two directed edges (v, v') and (v', v).

For a *directed* edge $(v, v') \in E$, let us define a *right cyclic walk* $\rho(v, v')$ as follows: Imagine a large castle in which the polygon traverses are wall. We stand (a bit removed from) the vertices v and put our left hand on the wall of the polygon traverse $\eta(v, v')$. Then we start walking forward, always touching the wall with our left hand. When we reach the vertex v', we continue walking and keep our left hand on the wall. Then we will be touching the wall linking v' to the next vertex v'' 'to the right' of v'. We continue our walk until we reach v once more. The sequence of visited vertices forms the right cyclic walk $\rho(v, v')$.

For a given right cyclic walk, let us define the notions of *inside* and *outside*. A point $p \in \mathbb{N} \times \mathbb{N}$ is *inside* the walk, if a ray shot from p, say, upwards, crosses the walk an odd number of times. Otherwise, the point is *outside* the walk. Note that it is not immediately clear, what counts as a 'crossing' of a ray and the walk: for example, the ray might only touch the walk or the walk might partly lie on it. To resolve these problems, let us imagine that the right cyclic walk is always 'slightly removed to the right' from the actual polygon traverse. In other words, not the wall itself is important, but where we walk. With this convention, a ray will always intersect a right cyclic walk 'cleanly'.

The algorithm for deciding reachability in planar-embedded undirected graphs works as follows: Given s and t, check for each directed edge $(v, v') \in E$, whether s is inside the right cyclic walk $\rho(v, v')$ and t is outside or whether t is inside this walk and s is outside. If either is the case for some edge, claim 'no path exists from s to t'. If neither is the case for all edges, claim 'there is a path from s to t'.

The algorithm needs only logarithmic space: A right cyclic walk can be constructed in logarithmic space and we can check in logarithmic space whether a point lies inside or outside such a walk.

To see that this algorithm yields correct outputs, first consider the case that there is a path from s to t in G. Then no right cyclic walk can separate s and t: on any path from s to t the vertices on the path would have to switch from 'inside' to 'outside' (or the other way round) at some point, which is impossible. Second, assume that there is no path from s to t in G. Consider the reachability components S and T of s and t. Since these components are disjoint, either they

must lie alongside each other, or S is 'inside' T, or T is 'inside' S. In the first case (the components lie alongside each other), consider an edge (v, v') on the outer border of S. Either $\rho(v, v')$ or $\rho(v', v)$ will be exactly this outer border. Then s is inside, but t is outside. For the second case, the outer border of S once more separates s and t. For the third case, the outer border of T separates them. □

3.3 Shortest Cycle Problems

In this subsection we study problems related to finding the smallest cycle in graphs. For directed graphs, this problem is fairly easy to classify, for undirected graphs its complexity remains largely unsolved.

SHORTEST-CYCLE (SC)
Instances. Directed graph $G = (V, E)$.
Solutions. Cyclic walk in G.
Measure. Length of the cyclic walk.

Theorem 3.17.

1. SC *is* $\leq_{\mathrm{E}}^{\log}$*-complete for* $\mathrm{NLO}_{\mathrm{pb}}$.
2. $\mathrm{SC} \in$ exp-ApxLO *iff* $\mathrm{SC} \in \mathrm{LO}$ *iff* $\mathrm{L} = \mathrm{NL}$.
3. $\mathrm{SC}_{\exists\mathrm{sol}}$ *and* $\mathrm{SC}_{\mathrm{opt}<}$ *are* $\leq_{\mathrm{m}}^{\log}$*-complete for* NL.

Proof. 1. We show $\mathrm{SP} \leq_{\mathrm{E}}^{\log} \mathrm{SC}$. The instance transformation function f gets an instance (G, s, t) for SP as input. We may assume that there exists a path from s to t and that the shortest such path has length at least 2. The function f first constructs a dag (a directed acyclic graph) as follows: let $V' := \{1, \ldots, n\} \times V$ and let there be an edge from (r_1, v_1) to (r_2, v_2) in E' iff $r_2 = r_1 + 1$ and $(v_1, v_2) \in E$. Then it adds further edges to E' as follows (thereby destroying the dag structure once more): for all $i \in \{1, \ldots, n\}$ let there be an edge from (i, t) to $(1, s)$. The resulting graph G' is the output of f.

The solution transformation function g gets a cyclic walk c in G' as input. Since G' was a dag before we added the special 'backward' edges, any such cyclic walk must pass through at least one such special edge. By 'rotating' the cyclic walk, we may assume that his edge is the first edge. Thus the cyclic walk starts with (x, t), $(1, s)$, $(2, v_2)$, $(3, v_3)$, and so on. Let (ℓ, v_ℓ) be the next vertex on the cyclic walk with $v_\ell = t$. The function g outputs the walk $(s, v_2, \ldots, v_\ell)$. Let $\alpha = 2$.

We must argue that the reduction is error preserving. To see this, note that the smallest cycle in G' is one edge longer (namely the edge leading back) than the shortest path in G from s to t. Let ℓ^* denote the length of the shortest cycle in G'. Then

$$\begin{aligned} \mathrm{R}_{\mathrm{SP}}\big(x, g(x, c)\big) - 1 &\leq \frac{\ell - 1}{\ell^* - 1} - 1 = \frac{\ell - \ell^*}{\ell^* - 1} = 2\frac{\ell - \ell^*}{\ell^* + \ell^* - 2} \leq 2\frac{\ell - \ell^*}{\ell^*} \\ &= \alpha\Big(\mathrm{R}_{\mathrm{SC}}\big(f(x), c\big) - 1\Big). \end{aligned}$$

2. As in Theorem 3.8, this follows from the third claim and from Theorem 4.3.

3. For the completeness of $\mathrm{SC}_{\exists\mathrm{sol}}$, which was first shown already in [7], just note that the graph G' constructed for the first claim has a cycle iff there is a path from s to t in G. □

The undirected version of the shortest cycle problem is much more mysterious. The little that is known about the problem is listed in Theorem 3.18.

UNDIRECTED-SHORTEST-CYCLE (USC)

Instances. Undirected graph $G = (V, E)$.

Solutions. Cyclic walk in G.

Measure. Length of the cyclic walk.

Theorem 3.18.

1. $\text{SC} \in \text{NLO}_{\text{pb}}$.
2. $\text{SC}_{\exists\text{sol}}$ *is* $\leq_{\text{m}}^{\text{AC}^0}$*-complete for* L.

Proof. The first claim is trivial, the second has been shown in [4]. □

3.4 The Hot Potato Problem

In this subsection we consider variants of the following problem: A hot potato (or, perhaps more realistically, a piece of news, a task, a packet, a token) must be passed around among a group of people for ℓ rounds. After ℓ rounds it will have cooled of. The 'value' (or 'cost') of getting rid of the potato by it to another person is given by a matrix and the objective is to maximize (or minimise) the total value (cost) of the path taken by the potato. Note that the potato is allowed to go round in cycles and may even be held by a person for any number of rounds (though, typically, this will not be attractive value-wise). Let us concentrate on the maximisation version. The same results also hold for the minimisation version.

In the following, the complexity of the problem and of different variants is studied. The basic hot potato problem has a logspace approximation scheme, but its budget problem is NL-complete. Two variants are especially intriguing: The budget problem of the undirected hot potato problem for matrices that have only two different entries is in L; but the same problem for matrices having three different entries is NL-complete.

MAXIMUM-HOT-POTATO-PROBLEM (MAX-HPP)

Instances. An $n \times n$ matrix A with entries drawn from $\{1, \ldots, n\}$, a number $\ell \leq n$, and a start index i_1.

Solutions. Index sequence $(i_1, \ldots, i_\ell)$.

Measure. Maximise $\text{s}_A(i_1, \ldots, i_\ell) := \sum_{p=1}^{\ell-1} A_{i_p, i_{p+1}}$.

Theorem 3.19.

1. *Optimal solutions for* MAX-HPP *can be computed in space* $O(\log \ell \log n)$.
2. MAX-HPP *is* $\leq_{\text{tt}}^{\log}$*-complete for* NLO_{pb}.
3. MAX-HPP $\in$ FLAS.
4. MAX-HPP $\in$ LO *iff* L = NL.
5. $\text{MAX-HPP}_{\exists\text{sol}} \in \text{AC}^0$, $\text{MAX-HPP}_{\text{opt}>}$ *is* $\leq_{\text{m}}^{\log}$*-complete for* NL.

Proof. 1. We argue similarly to the proof of Theorem 3.8. We use two algorithms reachable(u, v, c, ℓ) and construct(u, v, c, ℓ). The first algorithm will return true if there exists a sequence $\sigma = (u, j_1, \ldots, j_{\ell-1}, v)$ with $\text{s}_A(\sigma) \geq c$. Note that the length of this sequence is $\ell + 1$. The second algorithm will return such a sequence, provided it exists. Also, the second algorithm will omit the last element of the sequence.

procedure reachable(u, v, c, ℓ)
  let *flag* := false
  if $\ell = 1 \wedge A_{u,v} \geq c$ **then** *flag* := true

if $\ell > 1$ **then**
  forall $z \in V$ **do**
    forall $c' \in \{1, \ldots, c-1\}$ **do**
      if reachable$\big(u, z, c', \lfloor \ell/2 \rfloor\big) \wedge$ reachable$\big(z, v, c-c', \ell - \lfloor \ell/2 \rfloor\big)$ **then** *flag* := true
return *flag*

The algorithm needs space $O(\log \ell \log n)$, provided c is a polynomial in n.

procedure construct(u, v, c, ℓ)
  if $\ell \leq 1$ **then output** u
  else
    forall $z \in V$ **do**
      forall $c' \in \{1, \ldots, c-1\}$ **do**
      if reachable$\big(u, z, c', \lfloor \ell/2 \rfloor\big) \wedge$ reachable$\big(z, v, c-c', \ell - \lfloor \ell/2 \rfloor\big)$ **then**
        call construct$\big(u, z, c', \lfloor \ell/2 \rfloor\big)$
        call construct$\big(z, v, c-c', \ell - \lfloor \ell/2 \rfloor\big)$
        exit

This algorithm, too, needs only space $O(\log \ell \log n)$ provided c is a polynomial in n. The algorithm that produces optimal solutions works as follows: For each $c \in \{1, \ldots, n^2\}$ it tries to find an index i such that reachable$(i_1, i, c, \ell - 1)$ holds. For the largest c for which this is the case, it calls construct$(i_1, i, c, \ell - 1)$ for the corresponding i and then appends the missing index i.

2. We show SP $\leq_{\text{tt}}^{\log}$ MAX-HPP. Let (G, s, t) be an instance for SP. Let $n := |V|$. Our reduction maps this instance to n instances $(A^1, 2, 1)$, $(A^2, 3, 1)$, $\ldots$, $(A^n, n+1, 1)$ for MAX-HPP. Given optimal index sequences σ_1, $\ldots$, σ_n as answers to these queries, the reduction first discerns the length ℓ of the shortest path from s to t in G. Once it knows this length ℓ, it can retrieve a shortest path from s to t from the sequence σ_ℓ.

Let us start with the construction of the matrices A^i for $i \in \{1, \ldots, n\}$. We first construct a directed graph G' with $V' = \{1, \ldots, i+1\} \times V$ as follows. For rows $r_1, r_2 < i+1$, we insert edges roughly the same way as in Lemma 3.10: let there be an edge from (r_1, v_1) to (r_2, v_2) iff $r_1 = r_2 + 1$ and $(v_1, v_2) \in E \cup \{(v, v) \mid v \in V\}$. For the edges between row i and $i+1$ there is a special rule: there is just one edge from (i, t) to $(i+1, t)$. Note that this construction ensures that any walk in G' of length i must end at $(i+1, t)$.

Let us enumerate the elements of V' via a bijection $\nu \colon V' \to \{1, \ldots, |V'|\}$ with $\nu(1, s) = 1$. We construct an $|V'| \times |V'|$ matrix A^i as follows: let $A^i_{v'_1, v'_2} = 2$ if $(v'_1, v'_2) \in E'$ and let $A^i_{v'_1, v'_2} = 1$ if $(v'_1, v'_2) \notin E'$. Thus A^i is the adjacency matrix of the graph G', only with the modification that all entries are increased by 1.

The matrices A^i have the following property: there is a sequence $\sigma_i = (j_1, \ldots, j_{i+1})$ starting at $j_1 = 1$ with $\mathrm{s}_A(\sigma_i) \geq 2i$ iff there is a path from s to t in G of length at most $i-1$. To see this, first assume that such a sequence exists. Since all matrix entries are at most 2 and since the sequence has length $i+1$, we have $\mathrm{s}_A(\sigma) = 2i$. Thus we have $A_{j_p, j_{p+1}} = 2$ for all $p \in \{1, \ldots, i\}$. This means that the sequence $(v'_1, \ldots, v'_{i+1})$ with $\nu(v'_p) = j_p$ is a path in G'. Since the length of this path is i, as pointed out above, this path must end at $(i+1, t)$. This in turn means that there is a path from s to t in G of length at most $i-1$. This path can be obtained by removing the first components from each v'_p and omitting consecutive duplicates. Second, let us assume that there exists a path from s to t in G of length at most $i-1$. Then there exists a path $(v'_1, \ldots, v'_{i+1})$ from $(1, s)$ to $(i+1, t)$ in G'. Let $j_p := \nu(v'_p)$. Then for each $p \in \{1, \ldots, i\}$ we have $A_{j_p, j_{p+1}} = 2$. This in turn yields $\mathrm{s}_A(j_1, \ldots, j_{i+1}) = 2i$.

Suppose we are given optimal sequences $\sigma_1, \ldots, \sigma_n$ as answers to our queries. We determine the smallest ℓ such that $\mathrm{s}_A(\sigma_\ell) \geq 2\ell$. Then the shortest path from s to t in G has length $\ell - 1$. As shown above, we can obtain such a path from σ_ℓ.

3. For the logspace approximation scheme, first note that the problem is in $\mathrm{NLO_{pb}}$. This may not be obvious, since we have to compute the sum $\sum_{p=1}^{\ell-1} A_{i_p,i_{p+1}}$ while reading the sequence $(i_1, \ldots, i_\ell)$ in a one-way fashion. However, this can be done since all partial sums are bounded by n^2 and can thus be stored in logarithmic space.

The logspace approximation scheme for MAX-HPP works as follows: Let A, ℓ, and $r = 1+1/k$ be given. Let $\sigma_{\mathrm{opt}} = (i_1, \ldots, i_\ell)$ denote a sequence that maximises $\mathrm{s}_A(\sigma_{\mathrm{opt}})$. Our aim is to construct a sequence σ of length ℓ such that $\mathrm{s}_A(\sigma_{\mathrm{opt}})/\,\mathrm{s}_A(\sigma) \leq r = 1 + 1/k$.

For small values $\ell \leq 8k^2 + 10k + 4$ we perform a brute-force search to find the sequence σ_{opt}. By the first claim, this brute-force search takes space $O(\log k^2 \log n) = O(\log k \log n)$.

For large $\ell > 8k^2 + 10k + 4$ we find a sequence of $2k + 1$ indices $(j_1, \ldots, j_{2k+1})$ such that $c := \mathrm{s}_A(j_1, \ldots, j_{2k+1})$ is maximal. That is, we find a short walk of maximal value in the matrix. By iterating over all possible start points, it can be found in space $O(\log k \log n)$. We output the following sequence:

$$t := (\underbrace{i_1, \ldots, i_1}_{q \text{ times}}, j_1, j_2, \ldots, j_{2k+1}, j_1, j_2, \ldots, j_{2k+1}, \ldots, j_1, \ldots, j_{2k+1}),$$

where $q = 1 + (\ell - 1) \bmod (2k + 1)$. Then $\mathrm{s}_A(\sigma) \geq c\lfloor(\ell - 1)/(2k + 1)\rfloor$.

Now consider the sequence $\sigma_{\mathrm{opt}} = (i_1, \ldots, i_\ell)$. For each index $p \in \{1, \ldots, \ell - 2k\}$ we have $\mathrm{s}_A(i_p, i_p + 1, \ldots, i_p + 2k) \leq c$. To see this note that if this were not the case, then we would have $\mathrm{s}_A(i_p, i_p+1, \ldots, i_p+2k) > c$, which contradicts the optimality of the sequence $(j_1, \ldots, j_{2k+1})$. This shows $\mathrm{s}_A(\sigma_{\mathrm{opt}}) \leq c\lceil\ell/2k\rceil$.

We can now bound the performance ratio of the sequence σ as follows:

$$\begin{aligned}
\frac{\mathrm{s}_A(\sigma_{\mathrm{opt}})}{\mathrm{s}_A(\sigma)} &\leq \frac{c\lceil\ell/2k\rceil}{c\lfloor(\ell-1)/(2k+1)\rfloor} \leq \frac{\ell/2k+1}{(\ell-1)/(2k+1)-1} \\
&= \frac{(2k+1)(\ell+2k)}{2k(\ell-2-2k)} \leq \frac{(2k+1)(8k^2+10k+4+2k)}{2k(8k^2+10k+4-2-2k)} \\
&= \frac{(2k+1)(2k^2+3k+1)}{4k^3+4k^2+k} = \frac{(k+1)(4k^2+4k+1)}{k(4k^2+4k+1)} = 1 + \frac{1}{k}.
\end{aligned}$$

4. The only-if-part follows from the completeness of $\text{MAX-HPP}_{\text{opt}>}$, the if-part from Theorem 4.3.

5. For the completeness of the budget problem, we reduce GAP to $\text{MAX-HPP}_{\text{opt}>}$. Using the construction from the third claim, on input (G, s, t) we construct A^n and query whether $m^*(A^n, n+1, 1) \geq 2n$. This is the case iff the exists a path from s to t in G of length at most $n - 1$. □

UNDIRECTED-HOT-POTATO-PROBLEM (MAX-UHPP)

Instances. A symmetric $n \times n$ matrix A with entries drawn from $\{1, \ldots, n\}$, a number $\ell \leq n$, and a start index i_1.

Solutions. An index sequence $(i_1, \ldots, i_\ell)$.

Goal. Maximise $\mathrm{s}_A(i_1, \ldots, i_\ell) := \sum_{p=1}^{\ell-1} A_{i_p,i_{p+1}}$.

Theorem 3.20.

1. MAX-UHPP *is* $\leq_{\mathrm{tt}}^{\log}$*-complete for* $\mathrm{NLO_{pb}}$.

2. MAX-UHPP $\in$ FLAS.
3. MAX-UHPP $\in$ LO *iff* L = NL.
4. MAX-UHPP$_{\exists\text{sol}} \in \mathrm{AC}^0$, MAX-UHPP$_{\text{opt}>}$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* NL.

Proof. 1. We argue similarly to the proof of Theorem 3.19. However, there is a complication: we cannot simply take the symmetric closure of the graphs G' and then define the matrices A^i as in that proof. The reason is that an optimal sequence would then consist of constantly going back and forth along an edge of weight 2.

Rather, for each i, we construct the graph G' as the symmetric closure of the graph G' constructed in Theorem 3.19 and define the matrix A^i differently: For $r_1, r_2 < n+1$, if there is an edge $v_1' := (r_1, v_1)$ to $v_2' := (r_2, v_2)$ in G', let $A^i_{\nu(v_1'),\nu(v_2')} = i$. For the edge between $v_1' := (i, t)$ and $v_2' := (i+1, t)$ let $A_{\nu(v_1'),\nu(v_2')} = i+1$ and symmetrically $A_{\nu(v_2'),\nu(v_1')} = i+1$. All other entries of A are 1.

This time, there is a path from s to t in G of length at most $i-1$ iff there is a sequence $\sigma_i = (j_1, \dots, j_{i+1})$ staring at $j_1 = 1$ such that $\mathrm{s}_A(\sigma) \geq (i-1)i + (i+1) = i^2 + 1$. To see this, first let us assume that such a sequence exists. Since it has length $i+1$, we must have $A_{j_p,j_{p+1}} \geq i$ for all $p \in \{1, \dots, i\}$ and we must have $A_{j_p,j_{p+1}} \geq i+1$ at least once. This means that the sequence $v_1', \dots, v_{i+1}'$ with $\nu(v_p') = j_p$ is a path in G' that visits the edge between (i, t) and $(i+1, t)$. Thus there is a path from s to t in G. Second, let us assume that there exists a path from s to t in G of length at most $i-1$. Then for a path $(v_1', \dots, v_{i+1}')$ from $(1, s)$ to $(i+1, t)$ in G' and for $j_p := \nu(v_p')$ we have $\mathrm{s}_A(j_1, \dots, j_{i+1}) = (i-1)i + (i+1) = i^2 + 1$.

2. The approximation scheme for the directed problem also works here.

3. The only-if-part follows from the completeness of MAX-UHPP$_{\text{opt}>}$, the if-part from Theorem 4.3.

4. We reduce GAP to MAX-HPP$_{\text{opt}>}$, using the construction from the third claim. On input (G, s, t) we construct A^n and query whether $m^*(A^n, n+1, 1) \geq n^2 + 1$, which is the case iff the exists a path from s to t in G of length at most $n-1$. □

For a constant $k \geq 1$, let MAX-HPP-k-ENTRIES denote the problem MAX-HPP restricted to matrices that have only k different entries. Let MAX-HPP-ENTRIES-S denote the problem MAX-HPP restricted to matrices whose entries are drawn from the set $S \subseteq \mathbb{N}$. Define the undirected (or, if you prefer, symmetric) versions correspondingly.

The matrices constructed in Theorems 3.19 and 3.20 have only two, respectively three, different entries. In the non-symmetric case these entries are only 1 and 2. This proves the first of the following two theorems, which should be contrasted with each other.

Theorem 3.21. MAX-UHPP-3-ENTRIES$_{\text{opt}>}$ *and* MAX-HPP-ENTRIES-$\{1,2\}_{\text{opt}>}$ *are* $\leq_{\mathrm{m}}^{\log}$*-complete for* NL.

Theorem 3.22. MAX-UHPP-2-ENTRIES $\in$ LO *and* MAX-UHPP-ENTRIES-$S \in$ LO *for all finite* S.

Proof. For the first claim, let e_1 and e_2 with $e_1 \leq e_2$ be the two entry values found in the matrix A. If there exists an index i with $A_{1,i} = e_2$, then an optimal sequence is given by $(1, i, 1, i, \dots)$. If there does not exist any such index, let (i, j) be any pair with $A_{i,j} = e_2$. Then an optimal sequence is given by $(1, i, j, i, j, \dots)$.

For the second claim, let a symmetric matrix A be given as input with entries drawn from $S \subseteq \{1, \dots, k\}$ and let a length ℓ be given. For simplicity, let us assume that there is at least one pair (i, j) such that $A_{i,j} = k$. Let $\sigma_{\text{opt}} := (i_1, \dots, i_\ell)$ be a sequence that maximises $\mathrm{s}_A(\sigma_{\text{opt}})$. A lower bound on this value is given by $k(\ell - 2) \leq \mathrm{s}_A(1, i, j, i, j, \dots)$.

I claim that $A_{i_p,i_{p+1}} = k$ for all $p \in \{k+1, \dots, \ell-1\}$. To see this, suppose $A_{i_p,i_{p+1}} \leq k-1$ for all $p \in \{1, \dots, k+1\}$. Then $\mathrm{s}_A(i_1, \dots, i_{k+2}) \leq (k-1)(k+1) = k^2 - 1$. Since the remaining $A_{i_p,i_{p+1}}$ for $p \in \{k+2, \dots, \ell-1\}$ are bounded by k, we have $\mathrm{s}_A(\sigma) \leq k^2 - 1 + k(\ell - k - 2) = k\ell - 2k - 1$. However, we saw above that $\mathrm{s}_A(\sigma) \geq k\ell - 2k$.

In order to find an optimal sequence, we just have to find an optimal sequence of length $k+2$, which can be done in space $O(\log k \log n)$. Note that k is a constant. The last two entries i and j of this sequence will necessarily have the property $A_{i,j} = k$. Thus we can extend this sequence to an optimal sequence of length ℓ by appending the sequence $(i, j, i, j, \dots)$. □

3.5 Problems Related to Automata Theory

Problems related to automata theory are among the most difficult in NLO as was already noticed by Àlvarez and Jenner [1]. However, their focus lied on the completeness of the optimal measure functions for the function class OptL, rather than on the completeness of the optimisation problem itself for the class NLO.

MAX-WORD-NFA

Instances. Nondeterministic finite automaton M with ϵ-moves over the alphabet $\{0, 1\}$.

Solutions. Sequence of states leading from an initial state to an accepting state of length at most $|Q|$.

Measure. The word accepted along the state sequence, interpreted as a positive integer.

Theorem 3.23.

1. MAX-WORD-NFA *is* $\leq_{\mathrm{E}}^{\log}$*-complete for the class of maximisation problems in* NLO.
2. MAX-WORD-NFA *is* $\leq_{\text{1-tt}}^{\log}$*-complete for* NLO.
3. MAX-WORD-NFA $\notin$ exp-ApxLO*, unless* L = NL.
4. $\text{MAX-WORD-NFA}_{\exists\text{sol}}$ *and* $\text{MAX-WORD-NFA}_{\text{opt}>}$ *are* $\leq_{\mathrm{m}}^{\log}$*-complete for* NL.

Proof. 1. Let $P = (I, S, m, \max)$ be any problem in NLO. We construct a reduction (f, g, α) from P to MAX-WORD-NFA. Let M be the machine from Lemma 2.6.

The instance transformation function f maps an instance x to the following NFA: Its states are all configurations of M on input x. There is an ϵ-edge from one state c_1 to a state c_2 if M can go from the configuration c_1 to c_2 in one step without writing any output bit. There is an edge with a label i from c_1 to c_2 if M can go from c_1 to c_2 in one step and writes the bit i. The initial state of M is the start configuration, the set of final states is the set of M's accepting configuration.

The solution transformation function g maps a pair (x, p) consisting of an instance x and a walk p in M to a solution for x, by stringing together 'nondeterministic choices' taken on the walk p. Let $\alpha = 1$. By Lemma 2.6 we have $\mathrm{R}_P\big(x, g(x,p)\big) = \mathrm{R}_{\text{MAX-WORD-NFA}}\big(f(x), p\big)$.

2. By the first claim, we already know that the problem is $\leq_{\text{1-tt}}^{\log}$-complete for maximisation problems in NLO. We can also $\leq_{\text{1-tt}}^{\log}$-reduce minimisation problems $P = (I, S, m, \min)$ to MAX-WORD-NFA, namely as follows: We use the same construction as above, but with all edges with label 1 replaced by edges with label 0 and vice versa. Otherwise, the reduction is the same. With this construction, $m_{\text{MAX-WORD-NFA}}\big(f(x), p\big)$, read as a bitstring, is exactly the bitwise negation of $m\big(x, g(x,p)\big)$. Thus

$$m_{\text{MAX-WORD-NFA}}\big(f(x), p\big) = 2^\ell - 1 - m\big(x, g(x,p)\big),$$

where ℓ is the number from Lemma 2.6. This shows that for a solution p that maximises the left-hand side of the above equation, the value $m\big(x, g(x,p)\big)$ is minimal.

Note that this reduction does not preserve approximation ratios. For this we would have to *divide* $2^{2\ell}$ by $m\big(x, g(x,p)\big)$. It is not clear how that could be done, since we have only indirect access to this value.

3. and 4. We can easily reduce GAP to MAX-WORD-NFA$_{\exists\text{sol}}$. □

4 Structure of Logspace Optimisation and Approximation Classes

The previous section studied individual logspace optimisation problems and their individual approximation properties. In this section, where a more structural approach is taken, we study whether the logspace optimisation classes form proper hierarchies and whether they are closed under reductions.

4.1 Hierarchies of Logspace Optimisation Classes

The first observation about logspace approximation classes is that they form a proper hierarchy, unless L = NL. In the following theorem, we treat NLO, NLO$_{\text{do}}$, and NLO$_{\text{pb}}$ separately, since they do not match syntactically.

Theorem 4.1. *Suppose* L ≠ NL*. Then*

$$\begin{aligned}
&\text{LO} \subsetneq \text{FLAS} \subseteq \text{LAS} \subsetneq \text{ApxLO} \subsetneq \text{poly-ApxLO} \subsetneq \text{exp-ApxLO} \subsetneq \text{NLO},\\
&\text{LO}_{\text{do}} \subsetneq \text{FLAS}_{\text{do}} \subseteq \text{LAS}_{\text{do}} \subsetneq \text{ApxLO}_{\text{do}} \subsetneq \text{poly-ApxLO}_{\text{do}} \subsetneq \text{exp-ApxLO}_{\text{do}} \subsetneq \text{NLO}_{\text{do}},\\
&\text{LO}_{\text{pb}} \subsetneq \text{FLAS}_{\text{pb}} \subseteq \text{LAS}_{\text{pb}} \subsetneq \text{ApxLO}_{\text{pb}} \subsetneq \text{poly-ApxLO}_{\text{pb}} = \text{exp-ApxLO}_{\text{pb}} \subsetneq \text{NLO}_{\text{pb}}\,.
\end{aligned}$$

Proof. Suppose L ≠ NL. Then the claims following from the following non-inclusions, which we have already established in the theorems accompanying the individual problems.

1. TOURNAMENT-SP ∈ FLAS$_{\text{pb}}$, but TOURNAMENT-SP ∉ LO.
2. MVV-RATIO-2-PB ∈ ApxLO$_{\text{pb}}$, but MVV-RATIO-2-PB ∉ LAS.
3. MVV-PB ∈ pol-ApxLO$_{\text{pb}}$, but MVV-PB ∉ ApxLO.
4. MVV ∈ exp-ApxLO$_{\text{do}}$, but MVV ∉ poly-ApxLO.
5. SP ∈ NLO$_{\text{pb}}$, but SP ∉ exp-ApxLO.

□

The above theorem leaves one separation open: Does FLAS ⊊ LAS hold under the assumption L = NL? While the corresponding statement for polynomial time is known to hold, the argument used there does not carry over to logspace. However, we can at least prove a slightly weaker statement.

Theorem 4.2. *Suppose* NL ⊈ DSPACE$[f(n)\log n]$ *for some unbounded, monotone, logspace-computable function* f*. Then* FLAS ⊊ LAS*,* FLAS$_{\text{do}}$ ⊊ LAS$_{\text{do}}$*, and* FLAS$_{\text{pb}}$ ⊊ LAS$_{\text{pb}}$*.*

Proof. By Theorem 3.7, we have MVV-RATIO-DEC-PB ∈ LAS$_{\text{pb}}$, but MVV-RATIO-DEC-PB ∉ FLAS, unless NL ⊆ DSPACE$[\log^* n \log n]$. The function $\log^*$ was of no particular importance in the proof. By replacing it by f, we get the claim. □

We can also ask the inverse question: Does the hierarchy collapse if NL = L? In the polynomial-time setting, it is well-known that P = NP iff PO = NPO. For logarithmic space, the following theorem holds. While its only-if-part follows from the hierarchy theorem above, the more tricky if-part follows from a stronger lemma given below.

Theorem 4.3. *We have* $\mathrm{NLO}_{\mathrm{do}} \subseteq \mathrm{LO}$ *iff* $\mathrm{NL} = \mathrm{L}$.

Lemma 4.4. *For every* $P \in \mathrm{NLO}_{\mathrm{do}}$ *there exists a function in* $\mathrm{FL}^{\mathrm{NL}}$ *that outputs optimal solutions for* P.

Proof. We show that such a function exists for $P =$ PARTIAL-MVV. This proves the claim since PARTIAL-MVV is $\leq_{\mathrm{E}}^{\log}$-complete for $\mathrm{NLO}_{\mathrm{do}}$.

Given an instance (G, s, w) for PARTIAL-MVV, we construct an optimal path as follows: Using $\textsc{gap} \in \mathrm{NL}$ as an oracle, we find the most valuable vertex that is reachable from s. Having found such a vertex, using $\mathrm{SP}_{\mathrm{opt}<} \in \mathrm{NL}$ as an oracle, we construct a path from s to this vertex as follows: Starting from s, we check which successor of s is nearest to the target vertex, output this vertex, make this vertex out 'current vertex', and repeat until we have reached the target. □

Note that the implication 'if $\mathrm{NL} = \mathrm{L}$, then $\mathrm{NLO} = \mathrm{LO}$' is not known to hold.

4.2 Closure Properties of Logspace Optimisation Classes

If we wish to be formal, none of the classes introduced in this paper are closed under any of the reductions introduced in this paper. However, the reason for this is a bit annoying: Any optimisation problem (logspace or not) for which we can produce optimal solution in logarithmic space is $\leq_{\mathrm{E}}^{\log}$-reducible to all reasonable problems in $\mathrm{LO}_{\mathrm{pb}}$. However, there exist problems outside even NPO for which we can produce optimal solutions in logspace: their solution relation is made very hard (for example NEXP-complete), but we make the optimal solution always trivial.

The following theorems show that if we restrict ourselves to problems in NLO, the introduced classes enjoy the expected closure properties. For a reduction $\leq_r$ and a class C, let $\mathrm{R}_r(C) := \{P \mid P \leq_r P' \in C\}$ denote the *reduction closure* of C.

Theorem 4.5. $\mathrm{R}_{\mathrm{E}}^{\log}(C) \cap \mathrm{NLO} = C$ *for* $C \in \{\mathrm{LO}, \mathrm{FLAS}, \mathrm{LAS}, \mathrm{ApxLO}, \text{poly-ApxLO}, \text{exp-ApxLO}\}$.

Proof. Let $P \leq_{\mathrm{E}}^{\log} P' \in C$ via (f, g, α) and let $P \in \mathrm{NLO}$. Let $P' \in C$ via a approximator h, an approximation scheme h, or a function h the produces optimal solutions. Then $P \in C$ via a function e with $e(x) := g\big(x, h\big(f(x)\big)\big)$. The properties of E-reductions ensure that e has is a sufficiently good approximator, a sufficiently good approximation scheme, or a function that produces optimal solutions. □

Theorem 4.6. $\mathrm{R}_{\mathrm{AP}}^{\log}(C) \cap \mathrm{NLO} = C$ *for* $C \in \{\mathrm{ApxLO}, \text{poly-ApxLO}, \text{exp-ApxLO}\}$ *and* $\mathrm{R}_{\mathrm{AP}}^{\log}(\mathrm{LO}) \cap \mathrm{NLO} = \mathrm{LAS}$.

Proof. For the first claims, we can argue as in Theorem 4.5. For the last claim, let $P \in \mathrm{LAS}$ via an approximation scheme A. We show $P \leq_{\mathrm{AP}}^{\log} \textsc{trivial} \in \mathrm{LO}$, where all words are instances for TRIVIAL, exactly the word 0 is a solution for any instance, and all solutions have measure 1. The reduction works as follows: The instance transformation function f maps (x, r) to x. The solution transformation function g maps $(x, 0, r)$ to $A(x, r)$. Let $\alpha = 1$. □

Theorem 4.7. $\mathrm{R}_{\text{1-tt}}^{\log}(\mathrm{LO}) \cap \mathrm{NLO} = \mathrm{LO}$, $\mathrm{R}_{\text{1-tt}}^{\log}(\mathrm{LO}_{\mathrm{pb}}) \cap \mathrm{NLO}_{\mathrm{do}} = \mathrm{LO}_{\mathrm{do}}$, *and* $\mathrm{R}_{\text{1-tt}}^{\log}(\mathrm{FLAS}_{\mathrm{pb}}) \cap \mathrm{NLO}_{\mathrm{do}} = \mathrm{NLO}_{\mathrm{do}}$.

Proof. For the first claim, note that a $\leq_{\text{1-tt}}^{\log}$-reduction must map an optimal solution to an optimal solution.

For the second and third claim, we use the following observation: Every problem $P \in \mathrm{NLO}_{\mathrm{do}}$ is 1-tt-equivalent to a problem $P' \in \mathrm{NLO}_{\mathrm{pb}}$. The problem P' is defined as follows: It has the same

instance set and solution relation as P. The measure function m is defined differently: Let $x \in I$ be an instance. Let $M(x) = \{m_1, \ldots, m_\ell\}$ with $m_1 < m_2 < \cdots < m_\ell$ be the set of all outputs that could possibly be produced by m on input x. Such a set can be computed in logarithmic space since m produces its output deterministically. Thus we only need to cycle through all possible configurations in which m might start producing output. Define $m'(x, y) := i$ if $m(x, y) = m_i$. Then $P \equiv^{\log}_{1\text{-tt}} P'$.

For the second claim, we must show that any problem $P \in \mathrm{LO}_{\mathrm{do}}$ can be $\leq^{\log}_{1\text{-tt}}$-reduced to a problem $P' \in \mathrm{LO}_{\mathrm{pb}}$. By the above argument, $P \equiv^{\log}_{1\text{-tt}} P'$ for some $P' \in \mathrm{NLO}_{\mathrm{pb}}$. Since by the first claim $P' \leq^{\log}_{1\text{-tt}} P \in \mathrm{LO}$ implies $P' \in \mathrm{LO}$, we have $P' \in \mathrm{NLO}_{\mathrm{pb}} \cap \mathrm{LO} = \mathrm{LO}_{\mathrm{pb}}$.

For the third claim, note that TOURNAMENT-SP $\in$ FLAS is $\leq^{\log}_{1\text{-tt}}$-complete for $\mathrm{NLO}_{\mathrm{pb}}$ and that $\mathrm{R}^{\log}_{1\text{-tt}}(\mathrm{NLO}_{\mathrm{pb}}) \cap \mathrm{NLO}_{\mathrm{do}} = \mathrm{NLO}_{\mathrm{do}}$ by the above observation. □

4.3 Circuit Complexity of Logspace Optimisation Problems

In this subsection we establish that the circuit complexity of optimisation problems in NLO is as low as one might expect: we can compute optimal solutions using AC^1-circuits. This result extends a theorem of Àlvarez and Jenner [1], who showed that the optimal measure function (rather than an optimal solution function) of every problem in NLO is in FAC^1.

Theorem 4.8. *For every $P \in$ NLO there exists a function in logspace-uniform FAC^1 that computes optimal solutions for P.*

Proof. Let M denote the machine constructed in Lemma 2.6 for the a maximisation problem P. For an input word w, let G denote the reflexive closure of the configuration graph of M on input w. We assign labels to the edges of G as follows: Consider an edge from a configuration c_1 to a different configuration c_2. The labels are pairs (c, o) with $c \in \{0, 1, \epsilon\}$ and $o \in \{0, 1, \epsilon\}$. If c_2 is the only successor configuration of c_1, that is, if the step from c_1 to c_2 is deterministic, we set $c = \epsilon$. Otherwise we set $c = 0$ for the first successor c_2 of c_1 and $c = 1$ for the other successor c'_2 of c_1. We set o to be the output produced in the step from c_1 to c_2. If no output is produced, $o = \epsilon$. We assign the label (ϵ, ϵ) to all self-loops. The whole graph, including all labels, can be constructed in FAC^0.

We now introduce a 'pointer jumping operation' on directed graphs with labels that are pairs (c, o) with $c, o \in \{0, 1, \epsilon\}^*$. Given such a graph $G = (V, E)$, we define a graph $G' = (V, E')$ as follows: In G' there is an edge from u to v if there exists a vertex $z \in V$ such that $(u, z) \in E$ and $(z, v) \in E$. (This operation is known as pointer jumping.) We assign a label to the edge (u, v) as follows: for a given $z \in V$ with $(u, z) \in E$ and $(z, v) \in E$ let (c_z, o_z) be the label of the edge (u, z) and (c'_z, o'_z) of the edge (z, v). Pick z such that the concatenation $o_z o'_z$ is maximal lexicographically and define the label of (u, v) to be $(c_z c'_z, o_z o'_z)$. The pointer jumping operation can also be implemented in FAC^0 since concatenation and maximisation are in FAC^0.

The complete circuit works as follows: it constructs the graph G; applies the pointer jumping operation $\log n$ times, where n is the number of vertices in G; and outputs the c-part of the label of the edge from the initial configuration to the accepting configuration. This circuit is a AC^1-circuit.

It remains to show that the output is an optimal solution. To see this, note that the o-part of the label of the edge from the initial configuration to the accepting configuration is exactly the maximal output produced on any accepting computation of M. The o-part of this edge is exactly the string of non-deterministic choices taken along this path. By Lemma 2.6, this string is an optimal solution.

For minimisation problems we argue exactly the same way, replacing maximisation by minimisation. □

Corollary 4.9. NLO ⊆ PO.

5 Applications

5.1 Completeness in OptL

As mentioned in Section 2.1, there are two different ways of defining optimisation problems. The approach taken by Krentel [11] for polynomial time and by Àlvarez and Jenner [1] for logarithmic space considers optimisation problems to be much less structured objects. In their approach, an optimisation problem is just a function that maps an instance to the measure of an optimal solution.

Definition 5.1. The class OptL, respectively $\mathrm{OptL}_{\mathrm{do}}$, respectively $\mathrm{OptL}_{\mathrm{pb}}$, contains all functions $f\colon \Sigma^* \to \mathbb{N}$ such that there exists a logspace optimisation problem $P \in \mathrm{NLO}$, respectively $P \in \mathrm{OptL}_{\mathrm{do}}$, respectively $P \in \mathrm{OptL}_{\mathrm{pb}}$, with $f(x) = m^*(x)$ whenever $m^*(x)$ is defined and $f(x) = 0$ otherwise.

In a slight abuse of notation let us also write $g \in \mathrm{OptL}$ for partial functions $g\colon \Sigma^* \to \mathbb{N}^+$, meaning that $g' \in \mathrm{OptL}$ where $g'(x) = g(x)$ whenever $g(x)$ is defined and $g'(x) = 0$ otherwise.

Àlvarez and Jenner [1] define OptL slightly differently: they restrict it to maximisation problems. This restriction is confusing and should be avoided since completeness results are especially interesting if they hold for all of OptL, not just for the minimisation or maximisation part. The name MaxL, or perhaps max-OptL, seems better suited for the class treated by Àlvarez and Jenner. They also use slightly different denotations for the classes $\mathrm{OptL}_{\mathrm{do}}$ and $\mathrm{OptL}_{\mathrm{pb}}$ (restricted to maximisation problems), which they denote $\mathrm{OptL}\{\mathrm{determ}\}$ and $\mathrm{OptL}[\log n]$.

The completeness results established in this paper for logspace optimisation classes have direct corollaries in the form of new completeness results for OptL. Only the most interesting and surprising corollaries are stated.

Theorem 5.2. *The distance function for undirected graphs and the distance function for tournaments are* $\leq^{\log}_{1\text{-tt}}$*-complete for* $\mathrm{OptL}_{\mathrm{do}}$*. The functions* $m^*_{\text{MAX-HPP}}$ *and* $m^*_{\text{MAX-UHPP}}$ *are* $\leq^{\log}_{\mathrm{tt}}$*-complete for* $\mathrm{OptL}_{\mathrm{do}}$*.*

Proof. This follows directly from the corresponding completeness results for USP, TOURNAMENT-SP, MAX-HPP, and MAX-UHPP. □

5.2 Parallel versus Serial Queries

The results on the complexity of the undirected shortest path problem, have an interesting consequence for the question of whether parallel queries to UGAP can be simulated by serial queries.

Theorem 5.3. *If* $\mathrm{FL}^{\mathrm{SL}}_{\mathrm{tt}} \subseteq \mathrm{FL}^{X\,[\log]}$ *for some oracle* X*, then* $\mathrm{L} = \mathrm{SL}$*.*

Proof. By Theorem 3.9 there is a function $f \in \mathrm{FL}^{\mathrm{SL}}_{\mathrm{tt}}$ that maps an undirected graph G and two vertices s and t to a path from s to t, provided such a path exists. By assumption we have $f \in \mathrm{FL}^{X\,[\log]}$. Then f has an enumerator in FL. An *enumerator* [2] is a function that maps an input x to a set containing $f(x)$. We can then decide UGAP in logarithmic space, by applying the enumerator to the input graph and by checking whether at least one element in the enumerator's output is a walk from s to t. If so, we accept; otherwise, we reject. □

Corollary 5.4. $\mathrm{FL}_{\mathrm{tt}}^{\mathrm{SL}} = \mathrm{FL}^{\mathrm{SL}\,[\log]}$ *iff* $\mathrm{L} = \mathrm{SL}$.

The above corollary is a direct analogue to the equivalence '$\mathrm{FP}^{\mathrm{NP}} = \mathrm{FP}^{\mathrm{NP}\,[\log]}$ iff $\mathrm{P} = \mathrm{NP}$' due to Krentel [11] and to the equivalence '$\mathrm{FL}_{\mathrm{tt}}^{\mathrm{NL}} = \mathrm{FL}^{\mathrm{NL}\,[\log]}$ iff $\mathrm{L} = \mathrm{NL}$' due to Àlvarez and Jenner [1].

6 Conclusion

Different logspace optimisation problems can have vastly different properties, even though their underlying budget or existence problems have the same complexity. Research on logspace decision problems has been preoccupied with existence problems. The results of this paper suggest that we should broaden our perspective. Even in the context of decision problems, there are surprising differences between the complexity of existence problems and budget problems. A striking example is the shortest path problem for undirected graphs: while it is SL-complete to decide whether there *exists* a path from the first to the last vertex in an undirected graph, it is NL-complete to decide whether there exists such a path of a given maximal length.

We have seen that natural logspace optimisation problems can have different approximation properties. Most logspace optimisation problems are complete for an appropriate logspace approximation class or for NLO. Assuming $\mathrm{L} \neq \mathrm{NL}$, the approximation classes form a proper hierarchy and problems that are complete for one of these classes cannot be in any of the smaller classes.

The framework of this paper can be applied to many other problems that have already been studied in the literature. For example, Nisan, Szemeredi, and Wigderson [14] have given an algorithm for constructing a path from the first to the last vertex in an undirected graph in space $O(\log^{3/2} n)$. We may now ask whether anything can be said about *how long* this path will be, relative to the shortest path. We know that his path cannot be the shortest one unless $\mathrm{NL} \subseteq \mathrm{DSPACE}[\log^{3/2} n]$, but it might well be possible that USP can be approximated in space $O(\log^{3/2} n)$.

Another example is the reachability algorithm of Jakoby, Liśkiewitcz, and Reischuk [6] for series-parallel graphs. Their algorithm can be used to show that the reachability problem series-parallel graphs is in poly-ApxLO, but it is not clear at all how difficult it is to find the *shortest* path between two vertices in a series-parallel graph.

The most mysterious optimisation problem studied in this paper is the shortest undirected cycle problem. I would like to propose this problem for further research. Another interesting practical problem whose complexity is elusive is the shortest path problems for directed planar-embedded graphs.

References

[1] C. Àlvarez and B. Jenner. A very hard log-space counting class. *Theoretical Comput. Sci.*, 107:3–30, 1993.

[2] J.-Y. Cai and L. A. Hemachandra. Enumerative counting is hard. *Inf. Computation*, 82(1):34–44, 1989.

[3] A. Chiu, G. Davida, and B. Litow. Division in logspace-uniform NC^1. *Theoretical Informatics and Applications*, 35(3):259–275, 2001.

[4] S. Cook and P. McKenzie. Problems complete for deterministic logarithmic space. *J. Algorithms*, 8:385–394, 1987.

[5] P. Crescenzi, V. Kann, R. Silvestri, and L. Trevisan. Structure in approximation classes. *SIAM J. Computing*, 28(5):1750–1782, 1999.

[6] A. Jakoby, M. Liśkiewicz, and R. Reischuk. Space efficient algorithms for series-parallel graphs. In A. Ferreira and H. Reichel, editors, *Proc. 18th Annual Symp. on Theoretical Aspects of Comput. Sci.*, volume 2010 of *Lecture Notes on Comp. Sci.*, pages 339–352. Springer-Verlag, 2001.

[7] N. D. Jones. Space-bounded reducibility among combinatorial problems. *J. Comput. Syst. Sci.*, 11:65–85, 1975.

[8] D. R. Karger, R. Motwani, and G. D. S. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18(1):82–98, 1997.

[9] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science Volume A*, chapter 17, pages 869–941. The MIT Press/Elsevier, 1990.

[10] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On syntactic versus computational views of approximability. *SIAM J. Computing*, 28(1):164–191, 1998.

[11] M. W. Krentel. The complexity of optimization problems. *J. Comput. Syst. Sci.*, 36(3):490–509, June 1988.

[12] J. W. Moon. *Topics on Tournaments*. Holt, Rinehart, and Winston, 1968.

[13] A. Nickelsen and T. Tantau. On reachability in graphs with bounded independence number. In O. H. Ibarra and L. Zhang, editors, *Proc. 8th Int'l Computing and Combinatorics Conf.*, volume 2387 of *Lecture Notes on Comp. Sci.*, pages 554–563. Springer-Verlag, 2002.

[14] N. Nisan, E. Szemeredi, and A. Wigderson. Undirected connectivity in $O(log^{1.5}n)$ space. In *Proc. 33rd Annual Symp. on Foundations of Comput. Sci.*, pages 24–29, Pittsburgh, PA, 1992. IEEE.

[15] N. Nisan and A. Ta-Shma. Symmetric logspace is closed under complement. *Chicago J. Theoretical Comput. Sci.*, 1995(1), 1995.

[16] M. Ogihara and T. Tantau. On the reducibility of sets inside NP to sets with low information content. Technical Report TR-2002-782, Univ. of Rochester, Comput. Sci. Dept., www.cs.rochester.edu/trs/theory-trs.html, 2002.

[17] P. Orponen and H. Mannila. On approximation preserving reductions: Complete problems and robust measures. Technical Report C-1987-28, Univ. of Helsinki, Dept. of Comput. Sci., 1987.

[18] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

[19] T. Tantau. A note on the complexity of the reachability problem for tournaments. Technical Report TR01-092, Electronic Colloquium on Computational Complexity, www.eccc.uni-trier.de/eccc, 2001.